ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Hoàng Thị Điệp

CÁC PHƯƠNG PHÁP NHANH XÂY DỰNG CÂY BOOTSTRAP TIẾN HÓA

LUẬN ÁN TIẾN SĨ CÔNG NGHỆ THÔNG TIN

Hà Nội – 2019

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Hoàng Thị Điệp

CÁC PHƯƠNG PHÁP NHANH XÂY DỰNG CÂY BOOTSTRAP TIẾN HÓA

Chuyên ngành: Khoa học Máy tính
Mã số: 9480101.01

LUẬN ÁN TIẾN SĨ CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC:
1. PGS.TS. Lê Sỹ Vinh
2. PGS.TS. Hoàng Xuân Huấn

Hà Nội – 2019

Lời cam đoan

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi. Các kết quả được viết chung với các tác giả khác đều được sự đồng ý của các đồng tác giả trước khi đưa vào luận án. Các kết quả nêu trong luận án là trung thực và chưa từng được ai công bố trong các công trình nào khác.

Tác giả

Lời cảm ơn

Luận án được thực hiện tại Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội, dưới sự hướng dẫn của PGS.TS. Lê Sỹ Vinh, PGS.TS. Hoàng Xuân Huấn và TS. Bùi Quang Minh (hiện đang công tác tại Trung tâm Tin sinh Tích hợp Vienna, University of Vienna và Medical University Vienna, Vienna, nước Cộng hòa Áo).

Tôi xin bày tỏ lòng biết ơn sâu sắc tới PGS.TS. Hoàng Xuân Huấn, thầy đã giới thiệu cho tôi nhiều kiến thức bổ ích về toán và học máy thống kê và về nhiều bài toán ứng dụng khác nhau thông qua nhóm seminar học máy và tin sinh; giúp tôi định vị được bài toán của mình trong tổng thể. Thầy cũng đã nhiệt tình hướng dẫn tôi tìm hiểu một số bài toán tin sinh và tạo điều kiện cho tôi tham gia nhóm làm việc tại Viện nghiên cứu cao cấp về toán.

Tôi xin cảm ơn PGS.TS. Lê Sỹ Vinh, thầy đã tạo điều kiện tốt nhất để tôi kết nối với nhóm chuyên gia nghiên cứu ở Trung tâm Tin sinh Tích hợp Vienna; đồng thời luôn theo sát góp ý, lên kế hoạch, đốc thúc và động viên tôi làm nghiên cứu.

Tôi xin cảm ơn TS. Bùi Quang Minh, thầy đã giới thiệu cho tôi bài toán chính trong luận án này và hướng dẫn tôi vượt qua rất nhiều khó khăn khi triển khai các hướng giải quyết khác nhau cho bài toán, cũng như khi viết bài.

Tôi cũng xin cảm ơn tới các Thầy, Cô thuộc Khoa Công nghệ Thông tin, Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội đã tạo mọi điều kiện thuận lợi giúp tôi trong quá trình làm nghiên cứu sinh.

Cuối cùng, tôi xin gửi lời cảm ơn sâu sắc tới gia đình và bạn bè, những người đã cho tôi điểm tựa vững chắc để tôi hoàn thành tốt luận án này.

MỤC LỤC

Danh mục các ký hiệu và chữ viết tắt

IQPNNI	*thuật toán do Vinh và cộng sự* [49] *đề xuất để giải nhanh xây dựng cây tiến hóa theo tiêu chuẩn ML (Important Quartet Puzzling and NNI Optimization)*
ML	*tiêu chuẩn hợp lý nhất (Maximum Likelihood)*
MP	*tiêu chuẩn tiết kiệm nhất (Maximum Parsimony)*
MPBoot	*phương pháp mới luận án đề xuất để giải nhanh bài toán xây dựng cây bootstrap tiến hóa theo tiêu chuẩn MP*
MSA	*sắp hàng đa chuỗi (Multiple Sequence Alignment)*
NNI	*hoán đổi hàng xóm gần nhất (Nearest-Neighbor Interchange)*
RBS	*phương pháp bootstrap nhanh trong RAxML (RAxML Rapid Bootstrap)*
RELL	*lấy mẫu ước lượng log-likelihood (Resampling Estimated Log-Likelihoods)*
REPS	*lấy mẫu điểm MP (REsampling Parsimony Score)*
SBS	*phương pháp bootstrap chuẩn (Standard BootStrap)*
SPR	*cắt và ghép cây con (Subtree Pruning and Regrafting)*
TBR	*chặt đôi và nối lại (Tree Bisection and Reconnection).*
UFBoot	*phương pháp do Minh và cộng sự* [56] *đề xuất để giải nhanh bài toán xây dựng cây bootstrap tiến hóa theo tiêu chuẩn ML (UltraFast Bootstrap approximation)*
UFBoot2	*phương pháp luận án đề xuất để giải nhanh bài toán xây dựng cây bootstrap tiến hóa theo tiêu chuẩn ML*

UFBoot2+NNI	*thuật toán UFBoot2 tích hợp bước tinh chỉnh tối ưu để giảm ảnh hưởng của vi phạm mô hình*

Danh mục các bảng

Danh mục các hình vẽ, đồ thị

Danh mục các thuật toán

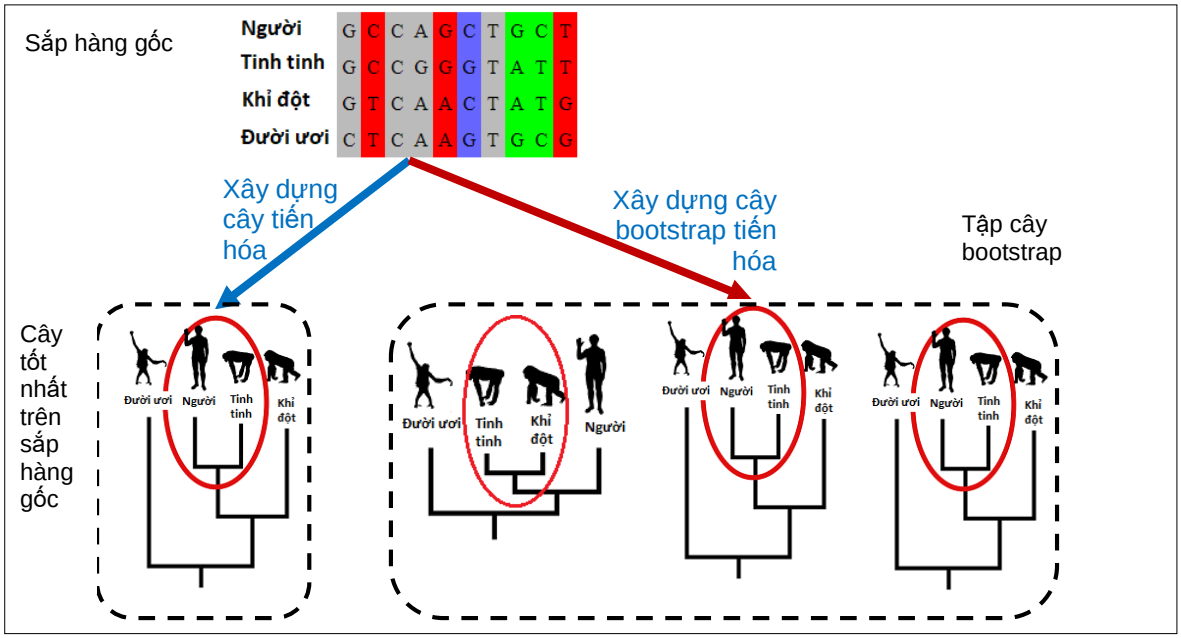
MỞ ĐẦU

Tái dựng lịch sử tiến hóa của sinh vật có ý nghĩa quan trọng trong lý giải các hiện tượng và góp phần thúc đẩy khám phá mới trong sinh học, y học và dược học. Ngày nay, nhờ phát triển mạnh của công nghệ giải trình tự DNA, việc tái dựng lịch sử tiến hóa thường tiến hành dựa trên dữ liệu sinh học phân tử. Lịch sử tiến hóa dựa trên dữ liệu sinh học phân tử còn cung cấp cơ sở cho các phương pháp phân tích hệ gen so sánh, với nhiều ứng dụng quan trọng trong đó nổi bật là tìm gen.

Theo học thuyết tiến hóa của Darwin tất các loài sinh vật đều tiến hóa từ một tổ tiên chung. Do đó, lịch sử tiến hóa thường có biểu diễn dạng cây với các lá đại diện cho các loài (còn gọi là taxa) – cũng có thể mở rộng cho các cá thể hoặc các gen – cần được phân tích quan hệ tiến hóa, còn gốc đại diện cho tổ tiên chung gần nhất của các taxa này. Cây tiến hóa được sử dụng trong nhiều lĩnh vực theo nhiều cách đa dạng. Chẳng hạn, trong nghiên cứu sự phát triển của virus cúm, do các biến đổi tiến hóa diễn ra liên tục, các chủng virus cúm mới được hình thành rất nhanh; dựa trên phân tích cây tiến hóa của chúng, các nhà khoa học có thể dự đoán chủng nào sẽ tuyệt chủng, chủng nào có khả năng cao tiếp tục phát triển để tạo thành dịch trong năm tiếp theo [8]. Một ví dụ khác, năm 1994, hai nhà khoa học Baker và Palumbi trong chuyến du lịch của mình tới Nhật Bản đã sử dụng bộ dụng cụ di truyền học để lấy mẫu thịt cá voi bán ở chợ và chứng minh rằng đây là loài cá voi lưng gù bị cấm săn bắt (theo Công ước quốc tế về quy định khai thác Cá Voi) thông qua việc phân tích cây tiến hóa [38]. Trong [64], phân tích tiến hóa được dùng để chứng minh việc một nha sĩ ở bang Florida, Hoa Kỳ đã lây nhiễm HIV cho các bệnh nhân của mình. Y học cũng đang hình thành một nhánh mới nhiều triển vọng được gọi là y học Darwin hay y học tiến hóa [58] với mục đích tìm hiểu và giải thích tại sao chúng ta mắc bệnh dựa trên phân tích tiến hóa.

Bài toán xây dựng cây tiến hóa nhận dữ liệu vào là sắp hàng của n chuỗi phân tử sinh học ứng với n loài và có mục tiêu xây dựng một cây nhị phân giúp giải thích

tốt nhất quá trình tiến hóa từ một tổ tiên chung thành n loài này (xem Hình 0.1).



Hình 0.1. Ví dụ minh họa đầu ra bài toán xây dựng cây tiến hóa và bài toán xây dựng cây bootstrap tiến hóa trong phân tích tiến hóa cho 4 loài.

Các phương pháp xây dựng cây tiến hóa có thể dựa vào khoảng cách hoặc dựa vào ký tự (còn gọi là vị trí sắp hàng) [103]. Các phương pháp dựa vào khoảng cách trước tiên tính khoảng cách tiến hóa cho từng cặp loài, sau đó sử dụng ma trận khoảng cách thu được để xây dựng cây. Khoảng cách tiến hóa giữa hai chuỗi là đại lượng tỉ lệ với số lượng biến đổi ký tự trạng thái thực sự xảy ra trong quá trình tiến hóa, được tính toán bằng cách kết hợp khoảng cách quan sát (là đại lượng tỉ lệ với số vị trí có ký tự trạng thái khác nhau trên hai chuỗi) và mô hình tiến hóa. Ví dụ: thuật toán ghép hàng xóm (neighbour joining – NJ) [71] áp dụng một thuật toán phân cụm trên ma trận khoảng cách tiến hóa để đưa ra một cây tiến hóa nhị phân. Các phương pháp dựa trên ký tự bao gồm các phương pháp tiết kiệm nhất (còn gọi là cực tiểu số lượng biến đổi, maximum parsimony – MP), các phương pháp hợp lý nhất (maximum likelihood – ML) và phương pháp suy luận Bayes. Các tiếp cận dựa trên ký tự tiến hành so sánh đồng thời tất cả các chuỗi trong sắp hàng, khảo sát lần lượt từng ký tự để tính điểm số cho một cây. Điểm số cho cây là số lượng biến đổi tối thiểu trong trường hợp MP, là giá trị log-likelihood trong trường hợp ML và là xác suất hậu nghiệm trong trường

hợp suy luận Bayes. Về lý thuyết, có thể xác định cây có điểm số tốt nhất bằng cách khảo sát và so sánh tất cả các cây ứng viên, chẳng hạn nhờ tìm kiếm vét cạn, nhưng điều đó lại không khả thi, trừ khi các bộ dữ liệu rất nhỏ. Do đó, ta cần sử dụng các thuật toán heuristic để tìm kiếm cây. Heuristic không đảm bảo tìm được cây tối ưu toàn cục (với một tiêu chuẩn tối ưu được chỉ định), nhưng nó có thể phân tích các bộ dữ liệu lớn. Để mô tả dữ liệu, các phương pháp dùng ma trận khoảng cách, ML và suy luận Bayes đều cần sử dụng mô hình tiến hóa và vì vậy còn gọi là phương pháp dựa trên mô hình, trong khi đó phương pháp MP không có mô hình rõ ràng và các giả thiết của nó ngầm ẩn. Luận án nghiên cứu việc phân tích dựa trên MP và ML, là điển hình của các tiếp cận này và được cộng đồng nghiên cứu quan tâm.

Phân tích bootstrap tiến hóa (hay *xây dựng cây bootstrap tiến hóa*) đề xuất bởi Felsenstein [21] là tiếp cận phổ biến để đánh giá mức độ tin cậy cho cây tiến hóa. Tiếp cận này được phát triển từ kỹ thuật bootstrap [15] của thống kê phục vụ việc xác định biến thiên của một ước lượng bằng thực nghiệm. Ngay sau khi xây dựng cây tiến hóa cho sắp hàng gốc, người ta thường tiến hành phân tích bootstrap phi tham số với thủ tục như sau: Với dữ liệu vào là sắp hàng gốc m vị trí, trước tiên, ta lấy mẫu có hoàn lại các vị trí trên sắp hàng gốc đúng m lần nhằm tạo một sắp hàng bootstrap (còn gọi là mẫu bootstrap hay bản sao bootstrap) có kích thước giống hệt sắp hàng gốc. Thông thường, 100 tới 1000 sắp hàng bootstrap được tạo ra theo cách này. Ký hiệu B là số sắp hàng bootstrap. Ta xây dựng cây tiến hóa cho mỗi sắp hàng bootstrap, gọi là cây bootstrap, theo cùng cách đã xây dựng cây tiến hóa cho sắp hàng gốc. Dữ liệu ra của bài toán làm phân tích bootstrap là tập cây bootstrap (xem minh họa ở Hình 0.1). Tập này thường được sử dụng như sau: gán cho mỗi cạnh trong của cây xây dựng trên sắp hàng gốc giá trị hỗ trợ bootstrap tính bằng tỷ lệ cây bootstrap có chứa cạnh đó.

Xây dựng tập cây bootstrap tiến hóa bằng cách tiến hành độc lập thuật toán xây dựng cây tiến hóa như nhau cho từng sắp hàng bootstrap được gọi là *phương pháp bootstrap chuẩn*. Theo đó, việc làm phân tích bootstrap với B bản sao bootstrap sẽ

tiêu tốn thời gian nhiều gấp *B* lần việc xây dựng cây tiến hóa. Bản thân bài toán xây dựng cây tiến hóa (mà không làm phân tích bootstrap) được xếp vào lớp NP-đầy đủ nếu sử dụng tiêu chuẩn MP [32] và NP-khó nếu sử dụng tiêu chuẩn ML [10]. Vì vậy, rút ngắn thời gian chạy luôn là vấn đề thách thức cho bài toán làm phân tích bootstrap. Vấn đề trở nên nghiêm trọng đặc biệt với sự ra đời của các công nghệ giải trình tự thế hệ tiếp theo (Next-generation sequencing – NGS) cho phép tạo ra những bộ dữ liệu khổng lồ. Một vấn đề lớn khác là giá trị hỗ trợ bootstrap tính bởi bootstrap chuẩn cho ta ước lượng thấp hơn xác suất đúng (tức xác suất thuộc về cây đúng) của cạnh [39,56]. *Cây đúng* là cây thể hiện đúng lịch sử tiến hóa với cấu trúc cây đúng và tất cả các độ dài cạnh đúng. Các phương pháp phân tích cây theo ML còn có các vấn đề về ảnh hưởng của vi phạm giả thiết mô hình (vi phạm mô hình) và ảnh hưởng của hiện tượng đa phân tới độ chuẩn xác bootstrap. Các tiếp cận nhanh đã được đề xuất để giải quyết vấn đề thời gian [5,33,36,45,56,79,81]. Trong số đó, UFBoot [56] là phương pháp nhanh nhất, được cài đặt trong hệ thống IQ-TREE (địa chỉ website: http://www.iqtree.org) và được sử dụng rộng rãi.

Từ đó, mục tiêu và kết quả luận án đã đạt được là:

1. Nghiên cứu phương pháp chuẩn và các phương pháp nhanh hiện tại cho xây dựng cây bootstrap tiến hóa theo tiêu chuẩn ML, đặc biệt là UFBoot, từ đó đưa ra đề xuất cải tiến để giải quyết tốt hơn từng thách thức của bài toán: thời gian chạy, độ chuẩn xác, ảnh hưởng của vi phạm mô hình và hiện tượng đa phân.
2. Nghiên cứu phương pháp chuẩn cho xây dựng cây bootstrap tiến hóa theo tiêu chuẩn MP, từ đó đề xuất phương pháp nhanh mới hiệu quả cho bài toán.
3. Cài đặt các đề xuất cải tiến cho UFBoot rồi tích hợp vào hệ thống IQ-TREE; xây dựng phần mềm MPBoot cài đặt phương pháp nhanh đề xuất cho MP để phục vụ nhu cầu của các nhà khoa học phân tích cây tiến hóa theo tiêu chuẩn ML và MP.

Các kết quả của luận án đã được công bố trong 2 bài báo ở tạp chí thuộc danh mục cơ sở dữ liệu của ISI (công trình khoa học số 2, 3) và 1 báo cáo ở hội nghị quốc tế (công trình khoa học số 1). Ngoài phần kết luận, luận án được tổ chức như sau:

Chương 1 trước tiên giới thiệu khái quát về các khái niệm cơ bản trong phân tích cây tiến hóa dựa trên dữ liệu sinh học phân tử: thông tin di truyền, sắp hàng đa chuỗi, cây tiến hóa, mô hình tiến hóa và các phương pháp xây dựng cây tiến hóa. Sau đó là phần giới thiệu về kỹ thuật bootstrap trong thống kê. Tiếp đó là phần phát biểu bài toán xây dựng cây bootstrap tiến hóa, các tiêu chí để đánh giá một phương pháp xây dựng cây bootstrap tiến hóa và các phương pháp hiện tại liên quan tới giải nhanh bài toán xây dựng cây bootstrap tiến hóa, trong đó tập trung vào UFBoot.

Chương 2 tập trung giải nhanh bài toán xây dựng cây bootstrap tiến hóa theo tiêu chuẩn ML. Chương này bắt đầu với lược đồ chung theo tiêu chuẩn ML; sau đó trình bày thuật toán pruning của Felsenstein tính likelihood cây và tóm tắt ý tưởng của phương pháp UFBoot giải nhanh bài toán bằng cách tìm một tập cây bootstrap chấp nhận được. Từ đây luận án đề xuất thuật toán pruning nhanh khi mô hình tiến hóa có tính thuận nghịch thời gian; sau đó trình bày đề xuất phương pháp UFBoot2 tích hợp thuật toán pruning nhanh và các kỹ thuật tối ưu mã nguồn để tăng tốc. Ngoài ra, luận án đề xuất thêm ba cải tiến quan trọng giải quyết các vấn đề về dữ liệu và mô hình mà UFBoot không hỗ trợ: (i) cải tiến để xử lý các đỉnh đa phân (ii) cải tiến để giảm ảnh hưởng của vi phạm mô hình và (iii) cải tiến mở rộng để phân tích sắp hàng nhiều gen. Kết quả thực nghiệm trình bày cuối chương đã chứng tỏ được hiệu quả của phiên bản nâng cấp này.

Chương 3 của luận án đề xuất một phương pháp MPBoot mới để tìm kiếm hiệu quả cây MP, đồng thời tìm nhanh lời giải chấp nhận được cho bài toán xây dựng cây bootstrap tiến hóa theo tiêu chuẩn MP. Kết quả thực nghiệm trên cả dữ liệu mô phỏng và các bộ dữ liệu sinh học lớn để so sánh MPBoot và phương pháp bootstrap chuẩn cài đặt trong hai công cụ phân tích theo tiêu chuẩn MP tốt nhất hiện nay là TNT và

PAUP* đã cho thấy MPBoot là lựa chọn thay thế tốt cho phương pháp bootstrap chuẩn theo tiêu chuẩn MP.

Chương 1 BÀI TOÁN XÂY DỰNG CÂY BOOTSTRAP TIẾN HÓA

1.1. Một số khái niệm cơ bản

Phần này của luận án sẽ trình bày các khái niệm cơ bản trong lĩnh vực phân tích quan hệ tiến hóa dựa trên dữ liệu sinh học phân tử: thông tin di truyền, sắp hàng đa chuỗi, cây tiến hóa.

1.1.1 Thông tin di truyền

Kiểu hình của sinh vật sống luôn là kết quả của thông tin di truyền (mà sinh vật mang và truyền cho thế hệ kế tiếp) và tương tác với môi trường. Vì vậy, để nghiên cứu về tiến hóa, ta cần tìm hiểu về biến đổi trong thông tin di truyền của sinh vật [50].

Bộ gen (genome) là tập hợp hoàn chỉnh các vật chất di truyền của một sinh vật [54]. Bộ gen của hầu hết các sinh vật là DNA (axít deoxyribonucleic), trong khi một số virus có bộ gen RNA (axít ribonucleic).

DNA là một phân tử mã hóa bản đồ thiết kế di truyền của sinh vật. Nói cách khác, DNA chứa tất cả các thông tin để tạo ra và duy trì một sinh vật. DNA được cấu tạo từ nhiều phân tử nhỏ gọi là các nucleotide. Có 4 loại nucleotide là: adenine (A), cytosine (C), guanine (G) và thymine (T) được xếp thành 2 nhóm purin (gồm A và G) và pyrimidine (gồm C và T) dựa trên các điểm tương đồng về cấu trúc hóa học. Các nucleotide kết hợp với nhau thành một mạch dài nhờ các liên kết hóa học để tạo thành một chuỗi nucleotide (còn gọi là chuỗi polynucleotide). DNA có cấu tạo gồm hai chuỗi nucleotide, trong đó các nucleotide giữa 2 chuỗi liên kết với nhau bằng liên kết hydrogen theo nguyên tắc bổ sung: A với T và G với C [1]. Do tính chất đặc biệt của ghép cặp bổ sung, mỗi chuỗi trong 2 chuỗi polynucleotide nói trên có chứa đầy đủ các thông tin mang trong chuỗi còn lại.

RNA được tổng hợp từ DNA nhờ quá trình phiên mã theo nguyên tắc bổ sung. Trong RNA, thymine được thay bằng uracil (U).

Axít amin là một hợp chất hữu cơ được cấu tạo bởi ba thành phần: nhóm amin ($-NH_2$), nhóm cacboxyl (-COOH) và nguyên tử cacbon trung tâm đính với 1 nguyên tử hyđrô và nhóm biến đổi R quyết định tính chất của axít amin [1,11]. Các axít amin kết hợp với nhau thành một mạch dài nhờ các liên kết peptide (còn gọi là chuỗi polypeptide) để tạo thành một chuỗi axít amin hay còn gọi là *protein*. Các chuỗi này có thể xoắn cuộn hoặc gấp theo nhiều cách để tạo thành các bậc cấu trúc không gian khác nhau của protein [2].

Mối quan hệ giữa nucleotide và axít amin được thể hiện qua quá trình tổng hợp protein. Trong một chuỗi nucleotide mã hóa protein, mỗi chuỗi ba nucleotide liên tiếp được gọi là một *codon*. Mỗi codon có thể mã hóa một axít amin hoặc là tín hiệu bắt đầu hoặc kết thúc của một quá trình tổng hợp protein [50]. Có tất cả 64 codon, trong đó có 61 codon mã hóa cho các axít amin, 3 codon còn lại được gọi là stop-codon (xem thêm Bảng 1.1).

Bảng 1.1. Danh sách 64 codon. Mỗi codon mã hoá một axít amin.

	T		**C**		**A**		**G**		
	Codon	**Axít amin**	**Codon**	**Axít amin**	**Codon**	**Axít amin**	**Codon**	**Axít amin**	
T	TTT	Phe	TCT	Ser	TAT	Tyr	TGT	Cys	**T**
	TTC	Phe	TCC	Ser	TAC	Tyr	TGC	Cys	**C**
	TTA	Leu	TCA	Ser	TAA	STOP	TGA	STOP	**A**
	TTG	Leu	TCG	Ser	TAG	STOP	TGG	Trp	**G**
C	CTT	Leu	CCT	Pro	CAT	His	CGT	Arg	**T**
	CTC	Leu	CCC	Pro	CAC	His	CGC	Arg	**C**
	CTA	Leu	CCA	Pro	CAA	Gln	CGA	Arg	**A**
	CTG	Leu	CCG	Pro	CAG	Gln	CGG	Arg	**G**
A	ATT	Ile	ACT	Thr	AAT	Asn	AGT	Ser	**T**
	ATC	Ile	ACC	Thr	AAC	Asn	AGC	Ser	**C**
	ATA	Ile	ACA	Thr	AAA	Lys	AGA	Arg	**A**
	ATG	Met	ACG	Thr	AAG	Lys	AGG	Arg	**G**
G	GTT	Val	GCT	Ala	GAT	Asp	GGT	Gly	**T**
	GTC	Val	GCC	Ala	GAC	Asp	GGC	Gly	**C**
	GTA	Val	GCA	Ala	GAA	Glu	GGA	Gly	**A**
	GTG	Val	GCG	Ala	GAG	Glu	GGG	Gly	**G**

Tuy nhiên, do có nhiều codon cùng mã hoá một axít amin nên số axít amin được mã hoá chỉ là 20 [11]. Tên đầy đủ và viết tắt của 20 axít amin được liệt kê đầy đủ trong Bảng 1.2.

Bảng 1.2. Danh sách 20 axít amin.

STT	Tên axít amin	Tên viết tắt (3 ký tự)	Tên viết tắt (1 ký tự)
1	Alanine	Ala	A
2	Arginine	Arg	R
3	Asparagine	Asn	N
4	Aspartic	Asp	D
5	Cysteine	Cys	C
6	Glutamine	Gln	Q
7	Glutamic	Glu	E
8	Glycine	Gly	G
9	Histidine	His	H
10	Isoleucine	Ile	I
11	Leucine	Leu	L
12	Lysine	Lys	K
13	Methionine	Met	M
14	Phenylalanine	Phe	F
15	Proline	Pro	P
16	Serine	Ser	S
17	Threonine	Thr	T
18	Tryptophan	Trp	W
19	Tyrosine	Tyr	Y
20	Valine	Val	V

1.1.2 Sắp hàng đa chuỗi

Để xây dựng cây tiến hóa, ta sử dụng các đặc tính tương đồng – là đặc tính ở những sinh vật khác nhau nhưng tương tự nhau bởi chúng được kế thừa từ một tổ tiên chung có đặc tính đó. Trong phân tích tiến hóa dựa trên dữ liệu phân tử sinh học, ta sử dụng các chuỗi nucleotide/axít amin tương đồng

Quá trình tiến hóa làm cho các chuỗi phân tử sinh học (nucleotide/axít amin) tương đồng khác nhau về nội dung cũng như độ dài. Sắp hàng đa chuỗi (multiple sequence alignment – MSA) sẽ giúp làm rõ các phép biến đổi giữa các chuỗi phân tử sinh học. Sắp hàng đa chuỗi có thể được hiểu như một ma trận các phân tử sinh học. Trong đó mỗi hàng chính là một chuỗi phân tử sinh học còn mỗi cột chứa các phân tử sinh học tương đồng của các chuỗi [50] (xem thêm Hình 1.1). Mỗi cột được gọi là một vị trí sắp hàng. Kích thước của một sắp hàng đa chuỗi được hiểu là số lượng chuỗi có trong sắp hàng đó, còn chiều dài của một sắp hàng đa chuỗi chính là chiều dài của một chuỗi trong sắp hàng. Hình 1.1 là một ví dụ của một sắp hàng đa chuỗi với bốn chuỗi axít amin của bốn loài linh trưởng. Sắp hàng có chiều dài là 15.

	1	**2**	**3**	**4**	**5**	**6**	**7**	8	9	10	11	12	13	14	15
Người	**E**	**H**	**D**	**-**	**N**	**D**	**E**	M	C	Q	L	K	P	L	P
Tinh tinh	**F**	**H**	**D**	**R**	**-**	**D**	**E**	M	C	Q	L	K	P	L	P
Khỉ đột	**F**	**G**	**R**	**R**	**-**	**D**	**E**	M	C	Q	L	K	P	L	P
Đười ươi	**F**	**G**	**D**	**R**	**-**	**V**	**H**	M	C	Q	L	K	P	L	P

Hình 1.1. Minh họa một sắp hàng đa chuỗi axít amin của bốn loài linh trưởng.

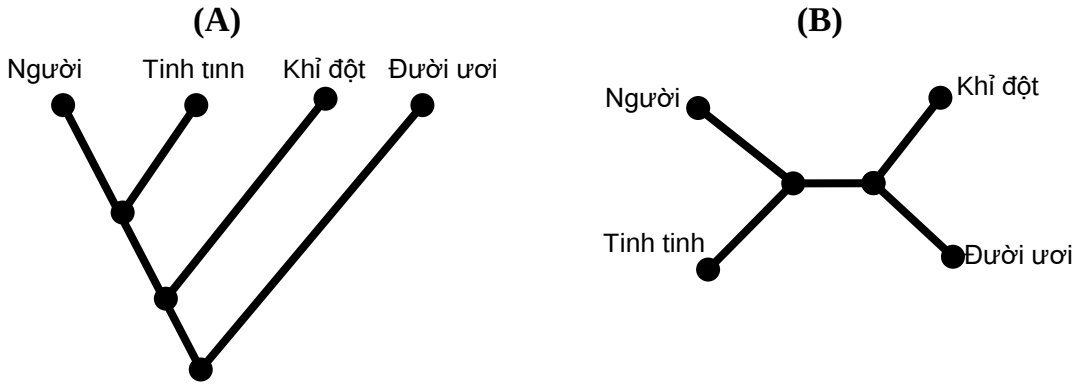
1.1.3 Cây tiến hóa

Cây tiến hóa là một dạng sơ đồ phân nhánh thể hiện mối quan hệ tiến hóa giữa các loài (cũng có thể mở rộng cho các cá thể hay các gen) dựa trên những điểm giống và khác trong thông tin di truyền. Các loài đưa vào phân tích quan hệ tiến hóa được cho là có một tổ tiên chung.

Mối quan hệ tiến hóa giữa các loài thường được biểu diễn bởi một cây nhị phân với cấu trúc như sau: mỗi đỉnh lá của cây biểu diễn một loài hiện tại (cho bởi dữ liệu vào); mỗi đỉnh trong của cây biểu diễn một loài tổ tiên (ta không có thông tin về các loài tổ tiên); mỗi cạnh của cây nối hai đỉnh của cây và biểu diễn mối quan hệ trực tiếp giữa hai loài ở hai đỉnh của cây; độ dài của cạnh nối hai loài trên cây cho biết khoảng cách tiến hóa giữa chúng. Khoảng cách tiến hóa này có thể được biểu diễn bằng thời

gian, hay số lượng các biến đổi nucleotide/axít amin giữa hai chuỗi nucleotide/axít amin được sử dụng để so sánh hai loài.

Ta thường không có thông tin về các loài tổ tiên, cho nên không xác định được chính xác gốc của cây tiến hóa. Chính vì vậy, cây tiến hóa thường được biểu diễn bằng một cây nhị phân không gốc như minh họa ở Hình 1.2B thay vì Hình 1.2A. Sau đó, khi có thêm thông tin tiến hóa, ta sẽ dùng một kỹ thuật định gốc, ví dụ như kỹ thuật outgroup: lúc đầu bổ sung vào sắp hàng một chuỗi nucleotide/axít amin của một loài khác xa với nhóm loài đang xét, tới cuối cùng khi đã tìm được cây thì đặt gốc trên cạnh nối với loài này [50].



Hình 1.2. Một ví dụ về cây tiến hóa giữa bốn loài linh trưởng: (A) dạng cây nhị phân có gốc và (B) dạng cây nhị phân không gốc.

Cây tiến hóa là một thành phần hữu ích trong các lĩnh vực sinh học, tin sinh học và nhân chủng học. Trong luận án này, nếu không có chú thích thêm thì cây tiến hóa được gọi tắt là cây. Ngoài ra, luận án gọi cây tiến hóa là cấu trúc phân nhánh khi muốn nhấn mạnh việc các cạnh của cây chưa xác định được độ dài mà chỉ thể hiện mối quan hệ liền kề giữa các đỉnh trong cây.

Luận án này sẽ bàn nhiều về *cạnh trong* của cây với vai trò phân hoạch. Đây là đơn vị nhỏ nhất trong quan hệ tiến hóa mà cây thể hiện. Xét cạnh trong $e(a, b)$ thuộc tập cạnh E của cây T, cạnh trong e sẽ phân tách tập các lá của cây T ra thành hai tập con $L_a|L_b$, trong đó tập L_a chứa các lá nằm cùng phía với đỉnh a; tập L_b chứa các lá nằm cùng phía với đỉnh b. Ta gọi $L_a|L_b$ là một cách phân hoạch của cây T. Lưu ý,

hai phân hoạch $L_a|L_b$và $L_b|L_a$ được coi là như nhau. Cây trong Hình 1.2B có duy nhất một cạnh trong. Nó thể hiện rằng "Người" có họ hàng gần với "Tinh tinh" hơn là với "Khỉ đột" và "Đười ươi".

1.2 Tổng quan phân tích tiến hóa

Theo Warnow [90], một quy trình phân tích quan hệ tiến hóa tổng quát và hoàn thiện gồm các bước:

(1) Nhà sinh học nhận định câu hỏi nghiên cứu, từ đó lựa chọn các loài và các gen cần phân tích.

(2) Thu thập chuỗi phân tử sinh học cho các loài và các gen này, thường là chuỗi DNA.

(3) Với mỗi gen, xây dựng sắp hàng đa chuỗi.

(4) Với mỗi gen, sử dụng sắp hàng đa chuỗi để xây dựng cây tiến hóa đơn gen.

(5) Tính toán độ hỗ trợ thống kê cho mỗi cạnh trong của cây tiến hóa đơn gen. Cách phổ biến nhất là dùng phương pháp bootstrap phi tham số (non-parametric bootstrap) [15]. Do trong luận án không khảo sát bootstrap có tham số (parametric bootstrap) [15] nên sau đây sẽ gọi tắt phương pháp bootstrap phi tham số là phương pháp bootstrap.

(6) Xây dựng cây tiến hóa hoặc mạng lưới tiến hóa đa gen (phylogenomic tree/network) nhờ kết hợp các cây tiến hóa đơn gen.

(7) Sử dụng kết quả của bước 6 để trả lời câu hỏi ở bước 1.

Bước 5 thường gắn chặt với kết quả của bước 4. Do đó, luận án tập trung vào bước 4 và 5 trong quy trình tổng thể này, tương đương với việc xây dựng cây tiến hóa có làm phân tích bootstrap.

1.3 Xây dựng cây tiến hóa

Cây tiến hóa không quan sát trực tiếp được mà phải suy luận từ chuỗi hoặc các dữ liệu khác. Bài toán xây dựng cây tiến hóa cho n loài từ n chuỗi sinh học đại diện cho n loài này là bài toán trung tâm của lĩnh vực phân tích tiến hóa phân tử sinh học.

Như đã bàn trong phần Mở đầu, luận án sẽ tập trung vào phân tích quan hệ tiến hóa theo tiêu chuẩn MP và ML do độ phổ dụng cao của chúng. Các phương pháp này dựa trên đầu vào là sắp hàng của n chuỗi sinh học đại diện cho n taxa, do đó tránh được vấn đề mất mát thông tin của các phương pháp khoảng cách. Ngoài ra, phân tích bằng MP và ML còn cho phép khôi phục chuỗi sinh học đại diện cho các loài tổ tiên ở trong cây.

1.3.1 Phát biểu bài toán

Dữ liệu vào: Dữ liệu đầu vào là một sắp hàng của n chuỗi phân tử sinh học (nucleotide/axít amin/codon) của n loài, mỗi chuỗi có m vị trí (ký tự). Với phân tích ML, dữ liệu vào có thêm mô hình tiến hóa.

Bài toán: Xây dựng cây tiến hóa biểu diễn mối quan hệ giữa n loài dựa vào phân tích sự giống nhau và khác nhau giữa các chuỗi nucleotide/axít amin của chúng.

Dữ liệu ra: Một cây nhị phân không gốc biểu diễn mối quan hệ giữa n loài và làm tối ưu hàm chấm điểm $f()$ theo tiêu chuẩn tối ưu được chỉ định. Cụ thể, cây tốt nhất sẽ làm cực tiểu *điểm MP* nếu sử dụng tiêu chuẩn MP, làm cực đại *điểm likelihood* nếu sử dụng tiêu chuẩn ML. Mỗi loài được biểu diễn ở một đỉnh lá của cây. Độ dài các cạnh của cây biểu diễn số lượng biến đổi ký tự trạng thái giữa các đỉnh của cây.

Theo đó, xây dựng cây tiến hóa có thể được xem là một bài toán tối ưu tổ hợp. Độ phức tạp về mặt thuật toán của bài toán này thể hiện ở số lượng lời giải ứng viên $S(n)$ - hay số lượng tất cả cấu trúc phân nhánh không gốc có thể có cho $n \geq 3$ loài cho bởi công thức:

$$S(n) = \prod_{i=3}^{n} (2i - 5)$$

Chú ý: với các cấu trúc phân nhánh đếm ở công thức trên, các cạnh của cây không chứa trọng số độ dài mà chỉ thể hiện mối quan hệ liền kề giữa các đỉnh trong cây và không có thông tin về các loài tổ tiên được biểu diễn bởi các đỉnh trong của cây.

Vì vậy, việc phát triển các chiến lược tìm kiếm heuristic cho bài toán này là cần thiết. Một việc quan trọng khác là tối ưu tính toán trong hàm chấm điểm $f()$ vì hàm này được gọi hàng triệu lần trong mỗi hàm tìm kiếm heuristic.

1.3.2 Tiêu chuẩn tiết kiệm nhất (maximum parsimony – MP)

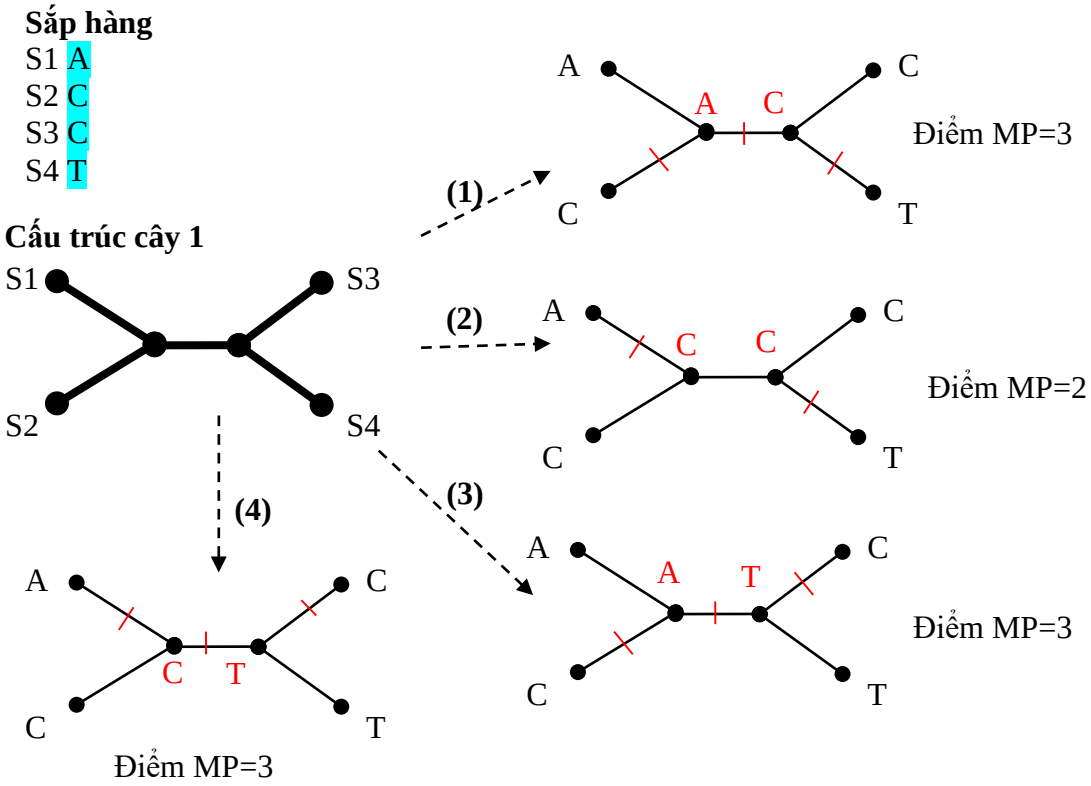
Theo tiêu chuẩn MP, cây tốt nhất là cây giải thích dữ liệu (tức sắp hàng đa chuỗi) bằng ít biến đổi nhất.

Điểm MP cho cây*:* Phương pháp MP tính số lượng biến đổi cực tiểu trên một cây tiến hóa bằng cách gán các ký tự trạng thái khác nhau cho các đỉnh trong của cây (xem ví dụ Hình 1.3). Điểm MP của cây trên một vị trí sắp hàng là số lượng biến đổi tối thiểu cần thiết cho vị trí đó. Điểm MP của cây trên toàn sắp hàng là tổng điểm MP trên tất cả các vị trí sắp hàng. Trong Hình 1.3, điểm MP cho cấu trúc cây 1 bằng 2.

Trong sắp hàng, vị trí chỉ chứa một loại ký tự trạng thái được gọi là vị trí hằng, nó luôn cho điểm MP bằng 0. Vị trí chứa hai loại ký tự trạng thái, trong đó một loại ký tự chỉ có trên duy nhất một loài được gọi là singleton. Singleton luôn cho điểm MP bằng 1. Vị trí hằng và vị trí singleton không có ích khi so sánh điểm MP của 2 cấu trúc cây nên thường bị bỏ đi.

Fitch [23] và Hartigan [35] đã đề xuất và phát triển một thuật toán tìm số lượng biến đổi tối thiểu cho 1 cây nhị phân (và xây dựng trạng thái tổ tiên để đạt cực tiểu). Tính toán điểm MP theo cách minh họa ở Hình 1.3 coi các biến đổi có chi phí như nhau. Nghĩa là, một cạnh có 2 đầu gán 2 ký tự khác nhau thì được tính điểm MP là 1,

ngược lại điểm MP là 0. Ta nói tính toán đó sử dụng *ma trận chi phí đều (uniform cost matrix)* (Bảng 1.3A).



Hình 1.3. Minh họa cách tìm điểm MP cho cấu trúc cây 1 bằng cách khảo sát 4 cách gán đỉnh trong.

Bảng 1.3. Ví dụ minh họa (A) ma trận chí phí đều và (B) ma trận chi phí không đều cho dữ liệu DNA.

(A) Ma trận chi phí đều

	A	C	G	T
A	0	1	1	1
C	1	0	1	1
G	1	1	0	1
T	1	1	1	0

(B) Ma trận chí phí không đều

	A	C	G	T
A	0	5	1	5
C	5	0	5	1
G	1	5	0	5
T	5	1	5	0

Trong thực tế tiến hóa của các phân tử sinh học, một số biến đổi xảy ra thường xuyên hơn một số biến đổi khác. Có thể xem các biến đổi xảy ra nhiều là có chi phí thấp hơn. Điều này có thể mô hình hóa trong tiêu chuẩn MP nhờ sử dụng *ma trận chi phí không đều (non-uniform cost matrix)* (Bảng 1.3B) và ta sử dụng thuật toán

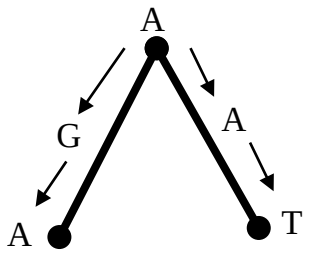
Sankoff [74] để tính điểm MP. Áp dụng ma trận ở Bảng 1.3B cho 4 cách gán đỉnh trong ở Hình 1.3, ta được điểm MP lần lượt là 11, 6, 11, 7. Do đó, điểm MP cho cấu trúc cây 1 là 6.

Cây MP là cây có điểm MP nhỏ nhất. Ta tìm cây này bằng cách duyệt và tính điểm MP cho tất cả các cấu trúc phân nhánh. PAUP* [84], MEGA [47] và TNT [31] là các chương trình phổ biến cho phân tích cây tiến hóa theo tiêu chuẩn MP.

1.3.3 Mô hình hóa quá trình biến đổi nucleotide

1.3.3.1 Đặt vấn đề

Khi phân tích cây tiến hóa theo tiêu chuẩn MP, điểm MP dựa trên khác biệt giữa các đỉnh liền kề. Nếu có đa biến đổi trong quá trình chuyển từ một trạng thái ở đỉnh cha sang một trạng thái ở đỉnh con (Hình 1.4), điểm MP không thể hiện chính xác số biến đổi thực sự đã diễn ra. Nhằm tính toán số lượng biến đổi thực sự, các phương pháp khoảng cách và các phương pháp dựa trên likelihood sử dụng mô hình xác suất để mô phỏng quá trình tiến hóa - sau đây gọi là *mô hình tiến hóa*.



Hình 1.4. Minh họa đa biến đổi trên cây gồm 1 đỉnh cha và 2 đỉnh con. Điểm MP bằng 1 trong khi số biến đổi thực sự là 3.

Mô hình tiến hóa giả sử rằng biến đổi tiến hóa tại một vị trí (nucleotide hay axít amin) tuân theo xích Markov thời gian liên tục với tập trạng thái $\mathcal{E}$ chính là tập các ký tự trạng thái. Mô hình Markov được sử dụng do nó mô phỏng được tính ngẫu nhiên và tính không nhớ của các biến đổi tiến hóa trong tự nhiên. Tức là, tại mỗi vị trí sắp hàng, một biến đổi tiến hóa xảy ra là ngẫu nhiên và trạng thái tiếp theo chỉ phụ thuộc trạng thái hiện tại chứ không phụ thuộc vào trạng thái trong quá khứ. Theo mô

hình này, m vị trí trong chuỗi của một loài tương đương với m xích Markov chạy dọc từ đỉnh gốc (loài tổ tiên) tới đỉnh lá (loài đang xét) tương ứng.

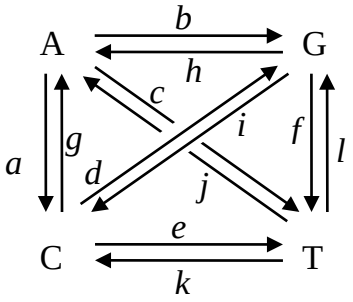
Phần này sẽ trình bày các công thức với mô hình biến đổi nucleotide (tập trạng thái $\mathcal{E} = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$); công thức cho mô hình biến đổi axít amin ($\mathcal{E}$ là tập các axít amin) hoạt động tương tự.

1.3.3.2 Ma trận tốc độ biến đổi tức thì

Tâm điểm của mô hình tiến hóa là một ma trận vuông $\mathbf{Q}$ kích thước $|\mathcal{E}| \times |\mathcal{E}|$. Trừ những ô nằm trên đường chéo, ma trận này thể hiện tốc độ biến đổi tức thì q_{ij} từ trạng thái ở hàng i sang trạng thái ở cột j. Dưới đây là ma trận tốc độ biến đổi tức thì $\mathbf{Q} = \{q_{ij}\}$ cho mô hình biến đổi nucleotide:

$$\mathbf{Q} = \begin{array}{c} \begin{matrix} \mathbf{A} & \mathbf{C} & \mathbf{G} & \mathbf{T} \end{matrix} \\ \begin{pmatrix} -\mu(a\pi_C + b\pi_G + c\pi_T) & \mu a\pi_C & \mu b\pi_G & \mu c\pi_T \\ \mu g\pi_A & -\mu(g\pi_A + d\pi_G + e\pi_T) & \mu d\pi_G & \mu e\pi_T \\ \mu h\pi_A & \mu i\pi_C & -\mu(h\pi_A + i\pi_C + f\pi_T) & \mu f\pi_T \\ \mu j\pi_A & \mu k\pi_C & \mu l\pi_G & -\mu(j\pi_A + k\pi_C + l\pi_G) \end{pmatrix} \end{array} \quad (1.1)$$

Trong đó, các hàng sắp theo thứ tự A,C,G,T. Ta gọi $\pi_\mathrm{A}, \pi_\mathrm{C}, \pi_\mathrm{G}, \pi_\mathrm{T}$ là tần suất của các nucleotide. Các tham số $a, b, c, d, e, f, g, h, i, j, k, l$ thể hiện tần suất tương đối (còn gọi là hệ số hoán đổi) của mỗi loại chuyển trạng thái (trong số 12 loại có thể xảy ra), trong suốt quá trình tiến hóa, giữa các ký tự trạng thái khác nhau (xem Hình 1.5). Ví dụ a thể hiện tần suất tương đối của biến đổi $\mathrm{A} \to \mathrm{C}$.



Hình 1.5. Tần suất tương đối của biến đổi giữa các nucleotide.

Gọi $\mathbf{P}(t)$ là ma trận xác suất chuyển giữa các ký tự trạng thái sau một khoảng thời gian tiến hóa t. Với giá trị $\Delta t > 0$ nhỏ, ma trận xác suất chuyển $\mathbf{P}(\Delta t)$ có thể được tính xấp xỉ tuyến tính theo khai triển Taylor như sau:

$$\mathbf{P}(\Delta t) \approx \mathbf{P}(0) + \mathbf{Q}\Delta t$$

Xét trạng thái i, để đảm bảo điều kiện tổng xác suất chuyển từ i đến các trạng thái khác bằng 1 sau một khoảng thời gian t bất kì, ma trận $\mathbf{Q}$ phải thỏa mãn điều kiện:

$$\sum_{j=1}^{|\mathcal{E}|} q_{ij} = 0$$

trong đó $q_{ij} \geq 0$ khi $i \neq j$.

Do đó, ta gán giá trị âm cho các ô đường chéo để đảm bảo tổng trong một hàng ma trận $\mathbf{Q}$ bằng 0.

$$q_{ii} = -\sum_{j \neq i} q_{ij}$$

Đại lượng μ là trung bình tốc độ biến đổi tức thì.

Trong tính toán, $\mathbf{Q}$ được chuẩn hóa để $-\sum_i \pi_i q_{ii} = 1$ (kéo theo $\mu = 1$). Khi ấy, độ dài cạnh của cây tiến hóa là kì vọng số lượng biến đổi nucleotide cho mỗi vị trí sắp hàng.

Mô hình tiến hóa chứa các giả thiết sau:

(1) Tại vị trí bất kỳ trên chuỗi, tốc độ biến đổi từ trạng thái i đến trạng thái j là độc lập với trạng thái trước i ở vị trí đó (tính chất Markov).

(2) Tốc độ biến đổi không đổi theo thời gian (tính đồng nhất).

(3) Tần suất tương đối của A, C, G và T ($\pi_{\mathrm{A}}, \pi_{\mathrm{C}}, \pi_{\mathrm{G}}, \pi_{\mathrm{T}}$) ở trạng thái cân bằng (tính dừng).

Trong Chương 2, luận án sẽ khảo sát một lớp cụ thể hơn của mô hình biến đổi nucleotide gọi là các mô hình có tính thuận nghịch thời gian (time-reversible), tức $a = g, b = h, c = j, d = i, e = k, f = l$.

Khi biết $\mathbf{Q}$ (theo đó, xác định được mô hình tiến hóa) ta có thể tính $p_{ij}(t)$, là xác suất chuyển từ trạng thái i sang trạng thái j sau thời gian tiến hóa t bằng cách tính hàm mũ ma trận:

$$\mathbf{P}(t) = \{p_{ij}(t)\} = e^{\mathbf{Q}t} \quad (1.2)$$

Ma trận xác suất chuyển trạng thái $\mathbf{P}(t)$ chính là chìa khóa để tính likelihood trong các phương pháp xây dựng cây tiến hóa theo ML.

1.3.3.3 Một số mô hình biến đổi nucleotide

Bảng 1.4 liệt kê các mô hình biến đổi nucleotide điển hình thuộc lớp thuận nghịch thời gian. Ta nhận thấy GTR [85,97] là mô hình tổng quát nhất với 8 tham số tự do. JC69 [43] là mô hình đơn giản nhất, nó không chứa một tham số tự do nào.

Bảng 1.4. Các tham số tự do của một số mô hình biến đổi nucleotide điển hình.

Tên mô hình	Các tham số tự do trong ma trận $\mathbf{Q}$
GTR (general time reversible)	8 tham số là : $a, b, c, d, e, \pi_A, \pi_C, \pi_G$ (vì $a + b + c + d + e + f = 1;\ \pi_A + \pi_C + \pi_G + \pi_T = 1$)
TN93	5 tham số là : $a = c = d = f, b = e, \pi_A, \pi_C, \pi_G$
HKY85	4 tham số là : $a = c = d = 1, b = e = k, \pi_A, \pi_C, \pi_G$
F84	Giống HKY85
F81	3 tham số là : π_A, π_C, π_G $(a = c = d = f = b = e)$
K80	1 tham số vì: $a = c = d = f = 1, b = e = k$ và $\pi_A = \pi_C = \pi_G = \pi_T$
JC69	0 tham số vì: $a = b = c = d = e = f$ và $\pi_A = \pi_C = \pi_G = \pi_T$

1.3.3.4 Tính không đồng nhất của tốc độ biến đổi giữa các vị trí trên chuỗi

Các mô hình trình bày ở phần trước đều giả sử các vị trí trong chuỗi có cùng tốc độ tiến hóa. Trong thực tế, tốc độ biến đổi nucleotide có thể khác biệt đáng kể ở các vị trí khác nhau trên chuỗi. Ví dụ, trong các gen mã hóa protein, các vị trí thứ 3 trong

codon thường biến đổi nhanh hơn các vị trí thứ nhất, các vị trí thứ 2 thì biến đổi chậm hơn 2 vị trí kia. Để tính tới tình huống này, cách phổ biến nhất là coi tốc độ biến đổi ở mỗi vị trí là một biến ngẫu nhiên có một phân bố thống kê nào đó [42]. Tiếp cận hay gặp nhất là phân bố gamma (+Γ) [96,98] với kì vọng bằng 1.0 và phương sai bằng $1/\alpha$ [50] do nó dễ diễn giải và nó khớp với dữ liệu thực [100], từ đó có các mô hình tiến hóa $JC69 + \Gamma$, $HKY85 + \Gamma$ hay $GTR + \Gamma$. Khi điều chỉnh tham số α, hình dạng của phân bố Γ có thể có dạng từ hình chuông ($\alpha > 1$) cho tới hình chữ L ($\alpha < 1$), do đó có thể biểu diễn được các mức độ khác biệt tốc độ khác nhau giữa các vị trí trên chuỗi. Để giảm chi phí tính toán, các phương pháp gần đây sử dụng mô hình gamma rời rạc, trong đó phân bố liên tục được xấp xỉ bởi k lớp tốc độ [98]. Bên trong mỗi lớp, tất cả các tốc độ sẽ được thay bằng kì vọng hoặc trung vị của lớp. Tuy không phổ biến như phân bố Γ, các phân bố khác cũng đã được khảo sát [87,102]. Cách tính likelihood khi mô hình có tính đến khả năng khác biệt tốc độ biến đổi giữa các vị trí được trình bày chi tiết trong [26].

1.3.4 Tiêu chuẩn hợp lý nhất (maximum likelihood – ML)

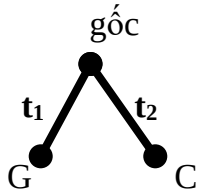
Cơ sở của tiếp cận hợp lý nhất: ML được phát triển bởi R. A. Fisher trong thập niên 1920 như một phương pháp luận thống kê để ước lượng các tham số chưa biết trong một mô hình. Hàm likelihood được định nghĩa là xác suất của dữ liệu khi biết các tham số nhưng lại thường được xem là hàm của các tham số khi dữ liệu đã quan sát được và cố định. Ước lượng hợp lý nhất là những giá trị của tham số làm cực đại likelihood. Thông thường, ước lượng hợp lý nhất được tìm thấy bằng các thuật toán tối ưu hóa lặp của phương pháp số.

Xây dựng cây tiến hóa theo ML: Thuật toán đầu tiên phân tích dữ liệu chuỗi DNA theo ML được Felsenstein phát triển [20]. Phương pháp này hiện đang được sử dụng rộng rãi nhờ vào sự gia tăng sức mạnh tính toán của máy tính và phần mềm và nhờ việc các mô hình tiến hóa ngày càng thực tế hơn. Lưu ý rằng để ước lượng cây tiến hóa theo ML cần hai bước tối ưu: (i) với một cấu trúc phân nhánh cụ thể, tối ưu

các tham số của mô hình tiến hóa và tối ưu độ dài các cạnh để tính điểm likelihood và (ii) khám phá không gian cấu trúc phân nhánh để tìm cấu trúc phân nhánh làm cực đại hàm likelihood. Suy luận cây theo ML tương đương với việc so sánh nhiều giả thuyết thống kê có cùng số lượng tham số.

Tính toán likelihood cho một cây cụ thể theo các mô hình biến đổi khác nhau được giải thích trong [22,101]. Do giả thiết về tính độc lập giữa tiến hóa của các vị trí trong chuỗi, *likelihood của cây trên sắp hàng* được tính bằng tích likelihood của cây cho từng vị trí.

Trong Hình 1.6, ta xét một ví dụ đã được đơn giản hóa để minh họa cách tính *likelihood của cây tại một vị trí sắp hàng* A_i, với tập trạng thái giả định $\mathcal{E} = \{\mathrm{G}, \mathrm{C}\}$, số lượng loài $n = 2$ và mô hình tiến hóa có các tham số đã xác định. Việc đầu tiên khi tìm likelihood của cây trên sắp hàng là đặt gốc trên một cạnh bất kì của cây. Trường hợp này, ta chỉ có một cách là đặt gốc trên cạnh nối 2 đỉnh lá, gọi cây ví dụ là cây T.



Hình 1.6. Một cây T đơn giản để minh họa cách tính likelihood của cây tại một vị trí sắp hàng.

Ta có likelihood của cây T tại A_i:

$$L(T|A_i) = P(A_i|T) = P(gốc = \mathrm{G}|T) + P(gốc = \mathrm{C}|T)$$
$$= P(gốc = \mathrm{G})\, p_{\mathrm{GG}}(t_1)\, p_{\mathrm{GC}}(t_2) + P(gốc = \mathrm{C})\, p_{\mathrm{CG}}(t_1)\, p_{\mathrm{CC}}(t_2)$$
$$\text{hay } L(T|A_i) = \pi_{\mathrm{G}}\, p_{\mathrm{GG}}(t_1)\, p_{\mathrm{GC}}(t_2) + \pi_{\mathrm{C}}\, p_{\mathrm{CG}}(t_1)\, p_{\mathrm{CC}}(t_2) \quad (1.3)$$

Trong (1.3), $\pi_{\mathrm{G}}, \pi_{\mathrm{C}}$ lần lượt là tần suất của các ký tự trạng thái G, C được cho trong mô hình tiến hóa; $p_{\mathrm{GG}}(t_1), p_{\mathrm{GC}}(t_2), p_{\mathrm{CG}}(t_1),\ p_{\mathrm{CC}}(t_2)$ được tính theo công thức (1.2) khi được cho mô hình tiến hóa.

Sau khi đã thiết lập được biểu thức của likelihood theo độ dài cạnh, ta điều chỉnh độ dài từng cạnh để làm cực đại likelihood. Với dữ liệu thực, việc khảo sát độ dài cạnh thường được thực hiện nhờ phương pháp số, chẳng hạn bằng phương pháp Newton-Raphson [63].

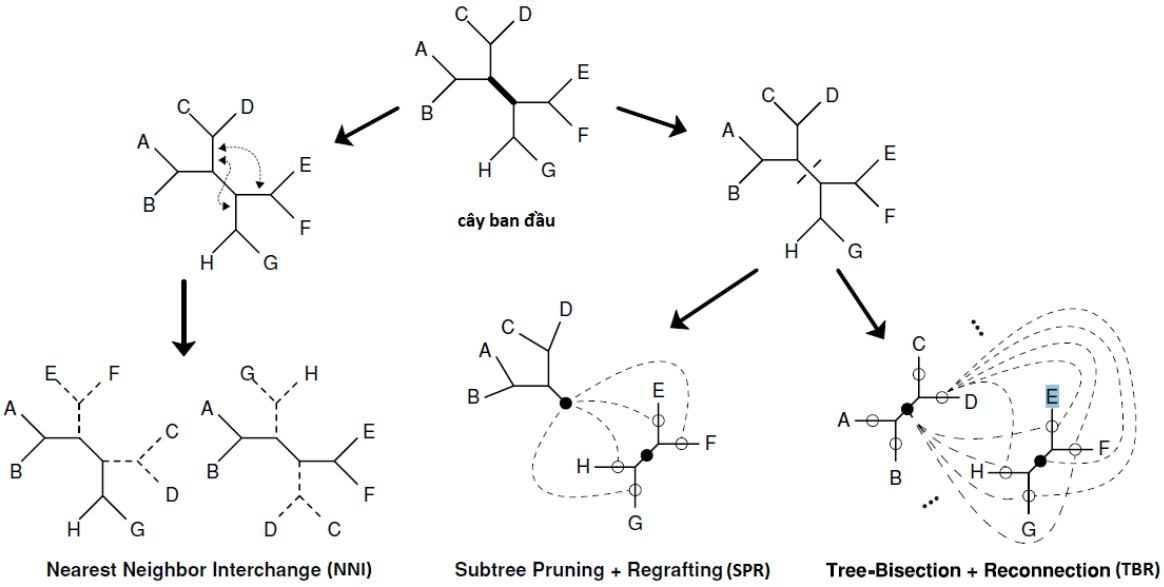
Các phần mềm ML hiện đại phân tích cây tiến hóa gồm có PhyML [34], RAxML [79], GARLI [105] và IQ-TREE [59]. Chúng không chỉ tính toán nhanh hơn mà còn hiệu quả hơn những phần mềm trước kia trong việc tìm kiếm cây có likelihood cao.

Việc tiêu chuẩn MP (không có mô hình tiến hóa rõ ràng) hay tiêu chuẩn ML (với một mô hình tiến hóa rõ ràng) là tốt hơn cho phân tích cây tiến hóa vẫn là chủ đề gây tranh cãi trong khoa học tiến hóa. Càng ngày tầm quan trọng của các phương pháp suy luận dựa trên mô hình càng được thừa nhận rộng rãi. Tuy nhiên, tiêu chuẩn MP vẫn được sử dụng: không phải do nó được cho là không cần giả thiết, mà do nó thường mang lại kết quả hợp lý, dễ hiểu và giải thích và nó hiệu quả về mặt tính toán.

1.3.5 Một số kỹ thuật biến đổi cục bộ trên cây dùng trong xây dựng cây tiến hóa

Do bài toán xây dựng cây tiến hóa theo tiêu chuẩn MP được xếp vào lớp NP-đầy đủ [32] còn theo tiêu chuẩn ML được xếp vào lớp NP-khó [10], tiếp cận phổ biến là sử dụng các phương pháp gần đúng. Để di chuyển trong không gian tìm kiếm (không gian cây), các phương pháp này thường có thủ tục leo đồi như sau: đầu tiên tạo một cấu trúc cây, sau đó biến đổi nó từng bước để cải thiện dần điểm số của cây theo tiêu chuẩn tối ưu được chỉ định. Có một lớp các kỹ thuật hay được dùng để cải thiện lời giải trong các thuật toán xây dựng cây tiến hóa có liên quan tới việc xáo trộn cấu trúc cây, còn được gọi là hoán-đổi-cạnh (branch-swapping) [50]. Các kỹ thuật này sẽ cắt một hay nhiều góc của cây (các cây con) và ghép chúng lại để được cây khác cục bộ so với cây ban đầu. Ba kỹ thuật hoán-đổi-cạnh (minh họa trong Hình 1.7 tham khảo từ [50]) từ đơn giản nhất đến phức tạp nhất là: (i) hoán đổi hàng xóm gần nhất (nearest-neighbor interchange – NNI), (ii) cắt và ghép cây con (subtree pruning

and regrafting – SPR) và (iii) chặt đôi và nối lại (tree bisection and reconnection – TBR). Lưu ý rằng trên một cây ban đầu, tập các xáo trộn NNI ứng viên là tập con của tập các xáo trộn SPR ứng viên; đây lại là tập con của tập các xáo trộn TBR ứng viên.



Hình 1.7. Ba kỹ thuật xáo trộn cấu trúc cây (NNI, SPR và TBR) trên cạnh tô đậm của cây ban đầu. Với SPR và TBR, tất cả các cặp cạnh đánh dấu bằng vòng tròn nhỏ trên 2 cây con sẽ được nối với nhau (các đường kẻ đứt), trừ phép nối 2 hình tròn đen với nhau vì nó sẽ tạo ra cây ban đầu. Nguồn: [50].

1.4 Giới thiệu phương pháp bootstrap trong thống kê

Bootstrap [15] là kỹ thuật trong thống kê để ước lượng theo cách thực nghiệm mức độ biến thiên của một ước lượng. Nó lấy mẫu có hoàn lại từ mẫu ban đầu để tạo ra một mẫu hư cấu có cùng kích thước.

Giả sử mẫu gốc có m điểm dữ liệu $\mathbf{x} = (x_1, x_2, \dots, x_m)$ được sinh độc lập từ phân bố $F(\theta)$ phụ thuộc vào tham số θ. Từ mẫu gốc ta tính được ước lượng $\hat{\theta} = t(x_1, x_2, \dots, x_m)$ cho tham số θ. Ta muốn biết mức độ biến thiên của phân bố của ước lượng này. Việc này trước khi phương pháp bootstrap ra đời là không thể thực hiện được nếu phân bố F chưa biết hoặc hàm ước lượng $t(\mathbf{x})$ phức tạp về mặt toán học [16]. Bootstrap suy luận ra biến thiên này bằng cách sử dụng mẫu gốc, thông qua việc sinh các mẫu mới không phải từ F mà từ phân bố thực nghiệm $\hat{F}$ cho các điểm dữ liệu trên mẫu gốc [22]. Sinh một mẫu cùng cỡ m từ phân bố $\hat{F}$ cũng giống như lấy

một mẫu các điểm $(x_1^*, x_2^*, \dots, x_m^*)$ từ chính mẫu gốc. Mẫu này được gọi là bản sao bootstrap $\mathbf{x}^*$. Từ bản sao ta cũng tính được ước lượng cho tham số $\widehat{\theta^*} = t(\mathbf{x}^*)$. Để thấy mức độ biến thiên của các ước lượng cho θ, ta chỉ cần sinh thật nhiều bản sao bootstrap và làm ước lượng trên đó. Các nghiên cứu đã chỉ ra rằng, m lớn và làm phân tích bootstrap với số lượng lớn bản sao sẽ cho biến thiên chính xác của các ước lượng cho θ.

Một ví dụ đơn giản minh họa việc sử dụng bootstrap trong thống kê là tìm khoảng tin cậy 95% của trung vị. Ta biết rằng không có lý thuyết thống kê nào cung cấp giải pháp để xác định khoảng tin cậy 95% của trung vị. Ta sẽ thực hiện bài toán này trên số liệu về điểm tổng kết môn xác suất thống kê của $m = 15$ sinh viên như sau:

Ví dụ minh họa

$$\mathbf{x} = (10, 4, 8, 5.5, 5, 3.5, 5, 4, 6, 8, 4, 4.5, 5, 7, 7)$$

Ta tìm được trung vị của mẫu gốc này là $\hat{\theta} = 5$.

Sinh mẫu bootstrap 1: Ta lấy mẫu có hoàn lại 15 lần các phần tử trong mẫu gốc để tạo mẫu bootstrap $\mathbf{x^*_1}$ và tính trung vị của nó.

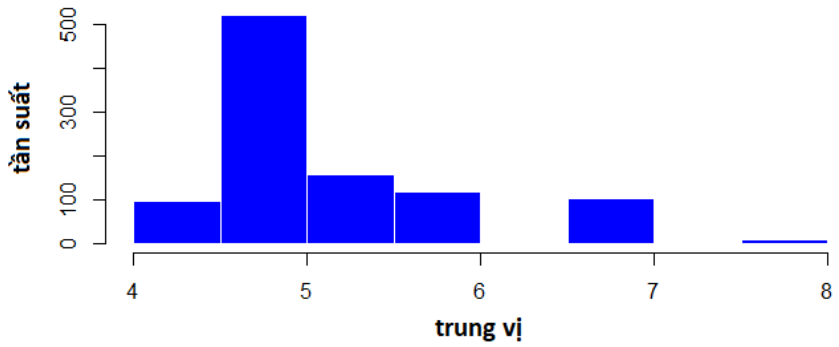
$$\mathbf{x^*_1} = (10, 5, 5.5, 4, 4.5, 7, 3.5, 7, 8, 6, 10, 4.5, 7, 4, 8);\ \widehat{\theta_1^*} = 6$$

Sinh mẫu bootstrap 2: Ta lấy mẫu có hoàn lại 15 lần các phần tử trong mẫu gốc để tạo mẫu bootstrap $\mathbf{x^*_2}$ và tính trung vị của nó.

$$\mathbf{x^*_2} = (7, 7, 3.5, 6, 5, 4, 5.5, 7, 4.5, 7, 6, 5.5, 4, 5, 5);\ \widehat{\theta_2^*} = 5.5$$

Sử dụng các phần mềm thống kê hiện đại (ví dụ RStudio cho phép phân tích số liệu bằng ngôn ngữ R), ta dễ dàng lặp lại 1000 lần việc sinh mẫu bootstrap như trên ($B = 1000$). Nhờ đó, ta thu được một phân bố của các trung vị bootstrap, thật ra là 1000 số trung vị. Sắp xếp các số này từ thấp đến cao. Chọn số ở hạng 2.5% và 97.5% của 1000 số trung vị. Đây chính là khoảng tin cậy 95%.

Kết quả của ví dụ này thực hiện trong RStudio: khoảng tin cậy 95% của trung vị là (4.5; 7) . Phân bố của $\widehat{\theta}^*$ được vẽ trong Hình 1.8.



Hình 1.8. Minh họa phân bố của trung vị mẫu tìm bằng phương pháp bootstrap.

1.5 Xây dựng cây bootstrap tiến hóa

1.5.1 Giới thiệu

Theo tổng thể phân tích tiến hóa trình bày trong phần 1.2, phương pháp bootstrap đề xuất bởi Felsenstein [21] có mục đích xác định độ hỗ trợ thống kê cho cây tốt nhất tìm được ở bước xây dựng cây tiến hóa. Ở đây, để vận dụng kỹ thuật bootstrap vào phân tích tiến hóa ta cần phải biểu diễn mẫu gốc (sắp hàng gốc) dưới dạng các điểm dữ liệu được lấy mẫu độc lập. Do không thể coi các loài là các điểm dữ liệu độc lập, mỗi vị trí sắp hàng sẽ là ứng viên tốt cho vai trò điểm dữ liệu.

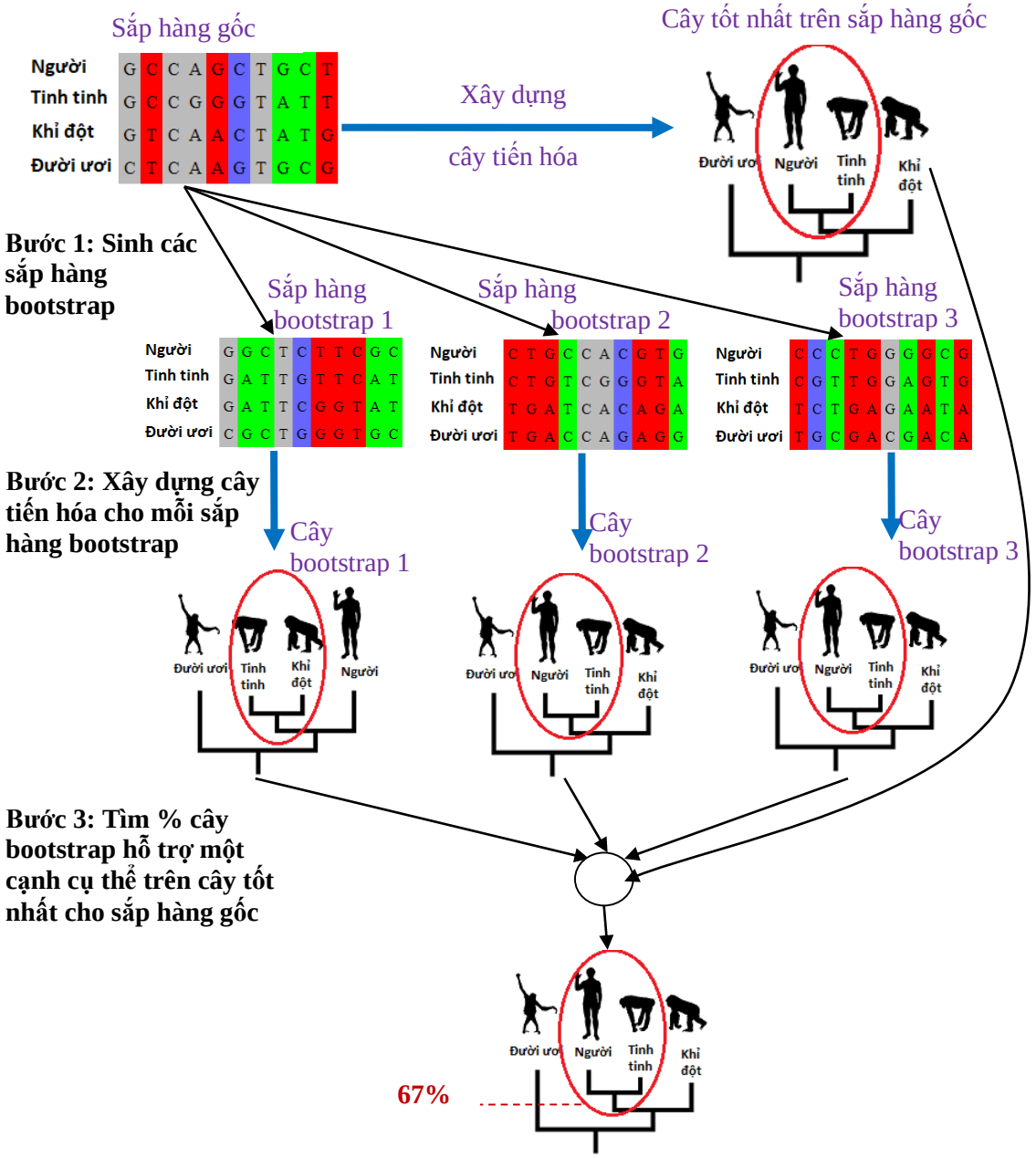
Phương pháp bootstrap chuẩn tiến hành như sau: Với sắp hàng gốc kích thước $n \times m$ (với n là số loài, m là số vị trí trong sắp hàng), lặp lại B lần việc sinh sắp hàng bootstrap có kích thước giống hệt sắp hàng mẫu bằng cách lấy mẫu có hoàn lại các vị trí của sắp hàng gốc m lần, ta thu được B sắp hàng bootstrap. Sau đó, ta xây dựng cây tiến hóa độc lập cho mỗi sắp hàng bootstrap (gọi là cây bootstrap) theo cùng cách đã xây dựng cây tiến hóa cho sắp hàng gốc. Thường thì sau khi thu được tập cây bootstrap, ta tiến hành thêm bước gán cho mỗi cạnh trong của cây xây dựng trên sắp hàng gốc giá trị hỗ trợ bootstrap tính bằng tỷ lệ cây bootstrap có chứa cạnh đó.

Quy trình làm phân tích bootstrap trong phân tích tiến hóa xuất phát từ một sắp hàng đa chuỗi được minh họa ở Hình 1.9. Theo kết quả minh họa thì, về mặt tiến hóa, "Người" gần với "Tinh tinh" hơn "Đười ươi" và "Khỉ đột". Giá trị hỗ trợ bootstrap của việc ghép nhóm (phân hoạch) này là 67%, tức là nhóm ("Người", "Tinh tinh") có mặt trong 67% tổng số cây bootstrap.

Một số khái niệm

Dưới đây là một số khái niệm liên quan tới bài toán xây dựng cây tiến hóa bootstrap.

- *Sắp hàng gốc* (original alignment) là dữ liệu vào (chuỗi sinh học cho n loài) dưới dạng ma trận $n \times m$ (mỗi loài ứng với một hàng, tức một chuỗi m ký tự) để xây dựng cây tiến hóa.
- *Sắp hàng bootstrap* (bootstrap alignment) là sắp hàng được sinh nhờ lấy mẫu không hoàn lại m lần các cột của sắp hàng gốc. Do đó, nó cũng là ma trận kích thước $n \times m$, mỗi hàng ứng với một loài.
- *Cây đúng* (true tree) là cây thể hiện đúng lịch sử tiến hóa với cấu trúc cây đúng và tất cả các độ dài cạnh đúng. Đây là cây giúp hình thành nên các chuỗi trong sắp hàng gốc từ một chuỗi tổ tiên chung. Trong thực nghiệm mô phỏng (ví dụ trong 1.5.3.2), cây đúng được chọn trước theo tiêu chí nào đó, sau đó được dùng để sinh sắp hàng gốc mô phỏng. Trong thực nghiệm với dữ liệu thực, ta không biết cây đúng.
- *Cạnh đúng* (true branch) là một phân hoạch thuộc về cây đúng.
- *Mô hình tiến hóa đúng* (true model) là mô hình giúp thể hiện đúng quá trình biến đổi giữa các ký tự trạng thái trong lịch sử tiến hóa để hình thành nên các chuỗi trong sắp hàng gốc từ một chuỗi tổ tiên chung. Giống với cây đúng, trong thực nghiệm mô phỏng, mô hình tiến hóa đúng được dùng để sinh sắp hàng gốc mô phỏng. Trong thực nghiệm với dữ liệu thực, ta không biết mô hình tiến hóa đúng.



Hình 1.9. Minh họa 3 bước làm bootstrap chuẩn phi tham số. Sắp hàng gốc có 4 taxa với 10 vị trí sắp hàng. Trong ví dụ này, ta làm bootstrap tiến hóa với 3 bản sao ($B = 3$). Phân tích thực tế thường cần tới 1000 bản sao bootstrap ($B = 1000$).

– *Cây tốt nhất trên sắp hàng gốc* (cây gốc, original tree) là kết quả của một thuật toán xây dựng cây tiến hóa theo một tiêu chuẩn tối ưu được chỉ định (MP hoặc

ML). Chất lượng của thuật toán sẽ quyết định cây này có gần với cây tối ưu toàn cục theo hàm chấm điểm (điểm MP hoặc likelihood) hay không.

- *Cây bootstrap tiến hóa* (cây bootstrap, bootstrap tree) là cây tốt nhất trên một sắp hàng bootstrap.
- *Giá trị hỗ trợ bootstrap* (bootstrap support value) là một số thuộc đoạn từ 0% đến 100% gắn vào một cạnh trong $e(a, b)$ (tương ứng với phân hoạch $L_a|L_b$) của cây gốc sau khi làm phân tích bootstrap. Nó được tính bằng phần trăm cây bootstrap có chứa phân hoạch này trong tập cây bootstrap.
- *Ước lượng bootstrap* (bootstrap estimate) là nói tới việc dùng kỹ thuật bootstrap như một cách để tìm ước lượng thống kê cho xác suất thuộc về cây đúng của mỗi cạnh trong. Theo đó giá trị hỗ trợ bootstrap là ước lượng cho xác suất thuộc về cây đúng của mỗi cạnh trong.
- *Độ chuẩn xác bootstrap* (bootstrap accuracy) hay *độ chuẩn xác của ước lượng bootstrap* (accuracy of bootstrap estimates) (xem Hình 1.10) là thước đo thể hiện sự khác biệt giữa kết quả ước lượng bootstrap và xác suất thuộc về cây đúng. Độ chuẩn xác bootstrap dùng trong việc đánh giá một phương pháp làm phân tích bootstrap. Một phương pháp cho kết quả chệch, tức luôn gán các giá trị hỗ trợ bootstrap cao hơn cho xác suất thuộc về cây đúng hoặc luôn gán các giá trị hỗ trợ bootstrap thấp hơn cho xác suất thuộc về cây đúng được xem là không chuẩn xác. Một phương pháp bootstrap chuẩn xác cần cho kết quả không chệch: nếu nó gán độ hỗ trợ bootstrap là 95% cho một cạnh thì cạnh đó phải có xác suất thuộc về cây đúng là 0.95. Ta phân tích độ chuẩn xác bootstrap nhờ thực nghiệm mô phỏng (xem chi tiết trong phần 1.5.3.2).
- *Phương pháp bootstrap chuẩn* (standard bootstrap, viết tắt là SBS) là phương pháp tìm ra tập cây bootstrap theo lược đồ chuẩn: khi được chỉ định số bản sao bootstrap là B, nó thực hiện B lần xây dựng cây tiến hóa sử dụng thuật toán giống nhau trên các sắp hàng bootstrap nhưng chạy độc lập với nhau.

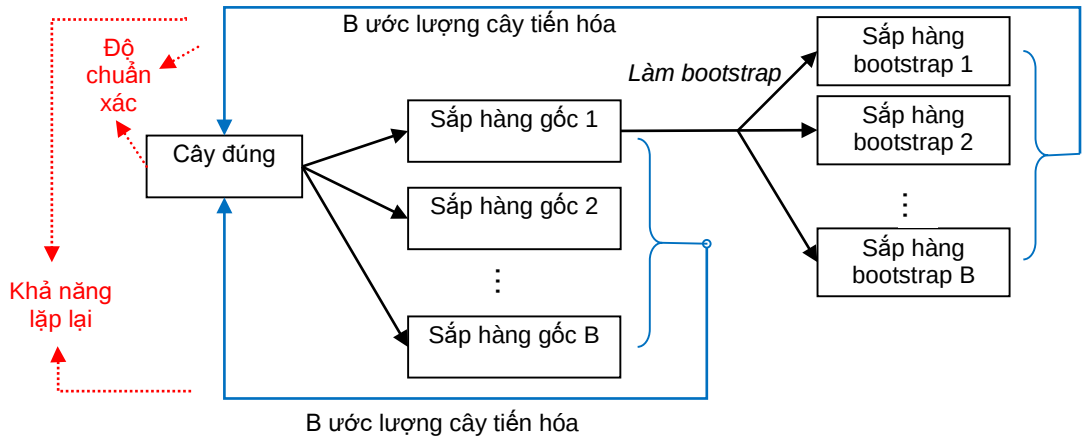
– *Phương pháp bootstrap nhanh* là một phương pháp cũng dựa vào B sắp hàng bootstrap để tìm ra tập cây bootstrap nhưng có thể không cần thực hiện độc lập B lần xây dựng cây tiến hóa sử dụng thuật toán giống nhau trên các sắp hàng bootstrap.

Ý nghĩa giá trị hỗ trợ bootstrap

Kết quả tìm khoảng tin cậy cho trung vị tìm được nhờ kỹ thuật bootstrap ở ví dụ phần 1.4 được hiểu là: nếu có thể lấy mẫu mới độc lập với mẫu gốc thì trung vị tính trên mẫu mới sẽ thuộc khoảng (4.5; 7) với xác suất 95%. Nghĩa là, số 95% là khả năng lặp lại (repeatability) của ước lượng. Trong ngữ cảnh phân tích tiến hóa, theo Hillis và Bull [39], mục đích ban đầu của Felsenstein [21] khi đề xuất làm bootstrap tiến hóa là nhằm đánh giá khả năng lặp lại (repeatability) của các cạnh trong cây tiến hóa (xem Hình 1.10).

Tuy nhiên, Hillis và Bull [39] đã tiến hành thực nghiệm và đi đến kết luận rằng: giá trị hỗ trợ bootstrap gán cho một cạnh trong e nên được hiểu là ước lượng cho xác suất để e thuộc về cây đúng. Nghĩa là, nếu cạnh trong $e(a, b)$ nối 2 đỉnh trong a và b được gán giá trị hỗ trợ bootstrap $x\%$ thì phân hoạch $L_a|L_b$ tương ứng có xác suất $x\%$ thuộc về cây đúng. Nghiên cứu của Hillis và Bull [39] cũng chỉ ra rằng dùng phương pháp bootstrap chuẩn thì ta thu được giá trị hỗ trợ bootstrap thấp hơn xác suất thuộc về cây đúng, vì vậy, một cạnh có độ hỗ trợ bootstrap 70% có thể coi là cạnh đúng. Đây cũng là cách hiểu được chấp nhận rộng rãi trong thực hành của khoa học tiến hóa.

Như vậy, khi nói tới độ chuẩn xác của phương pháp bootstrap là ta so sánh tập cây bootstrap với cây đúng. Còn khi nói tới khả năng lặp lại của phương pháp bootstrap là ta so sánh tập cây bootstrap với một tập cây tiến hóa giả tưởng ta sẽ xây dựng được nếu có thể thu thập thêm sắp hàng gốc tiến hóa từ cùng một cây đúng.



Hình 1.10. Minh họa khái niệm độ chuẩn xác và khả năng lặp lại khi làm bootstrap với B bản sao trên sắp hàng gốc 1.

1.5.2 Phát biểu bài toán

Dưới đây là phát biểu bài toán xây dựng cây tiến hóa có làm phân tích bootstrap.

Dữ liệu vào: Dữ liệu đầu vào là một sắp hàng của n chuỗi phân tử sinh học (nucleotide/axít amin/codon) của n loài, mỗi chuỗi có m vị trí (ký tự), sau đây gọi là sắp hàng gốc. Với phân tích ML, dữ liệu vào có thêm mô hình tiến hóa.

Bài toán: Sinh B sắp hàng bootstrap. Theo tiêu chuẩn tối ưu đã xác định (MP hoặc ML), xây dựng cây tiến hóa cho sắp hàng gốc và cho từng sắp hàng bootstrap. Do có nhiều phương pháp khác nhau cho kết quả với độ chuẩn xác cũng như thời gian thực hiện khác nhau, chúng ta cần đề xuất các phương pháp cho kết quả chuẩn xác với thời gian thực hiện thấp nhất có thể.

Dữ liệu ra: Cây tiến hóa tốt nhất cho sắp hàng gốc và tập cây bootstrap. Thường thì tập cây bootstrap sẽ được dùng để tính giá trị hỗ trợ bootstrap cho mỗi cạnh trong (tức phân hoạch, xem 1.1.3) của cây tốt nhất cho sắp hàng gốc.

1.5.3 Các tiêu chí đánh giá

Phần này trình bày hai tiêu chí quan trọng nhất khi đánh giá một phương pháp xây dựng cây bootstrap tiến hóa.

1.5.3.1 Đánh giá thời gian chạy

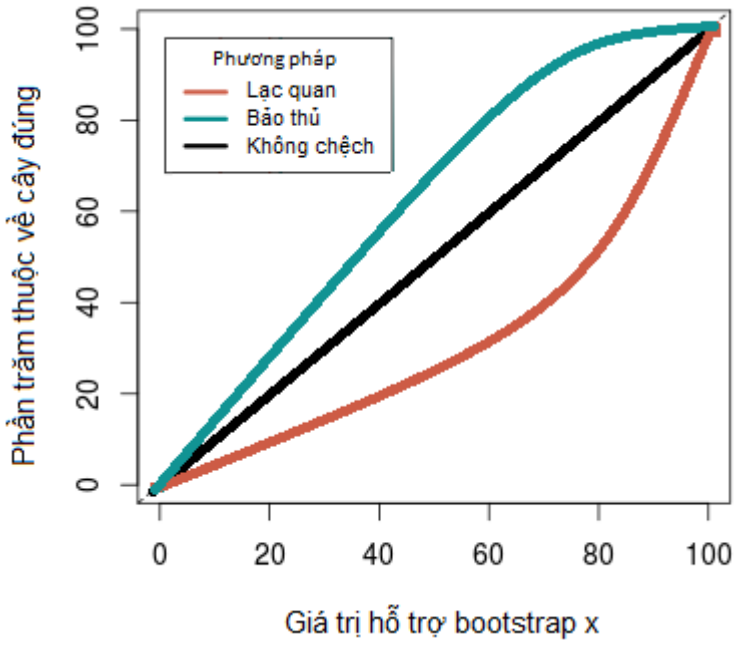
Đây là đánh giá đơn giản nhất. Ta đo thời gian chạy của từng phương pháp xây dựng cây bootstrap tiến hóa trên cùng dữ liệu vào trên các máy tính có cùng cấu hình. Do việc làm phân tích bootstrap thường đi kèm với việc xây dựng cây (duyệt tìm cây tốt nhất) cho sắp hàng gốc nên trong luận án này, thời gian khảo sát là tổng của thời gian xây dựng cây cho sắp hàng gốc và thời gian làm phân tích bootstrap.

1.5.3.2 Đánh giá độ chuẩn xác của ước lượng bootstrap

Việc khảo sát độ chuẩn xác của phương pháp bootstrap Z có thể được thực hiện thông qua thực nghiệm mô phỏng. Từ một cây đúng $T^{đúng}$ ban đầu và một mô hình tiến hóa M, ta sinh ra K sắp hàng mô phỏng bằng công cụ mô phỏng tiến hóa nào đó. Luận án sử dụng công cụ Seq-Gen [69] - là công cụ mô phỏng tiến hóa tin cậy nhất hiện nay. Với mỗi sắp hàng mô phỏng A_k , ta xây dựng cây tiến hóa $T_k^{tốt\ nhất}$, rồi dùng phương pháp Z để làm phân tích bootstrap (với B bản sao bootstrap) nhằm tính giá trị hỗ trợ bootstrap cho từng cạnh trên $T_k^{tốt\ nhất}$. Từ các cạnh của tập K cây $T_k^{tốt\ nhất}$, ta tính tỉ lệ các cạnh thuộc cây đúng $T^{đúng}$ trong số tất cả các cạnh có giá trị hỗ trợ bootstrap $x\%$ với $x = 0, \dots, 100$.

Độ chuẩn xác của một phương pháp, Z, được định nghĩa bởi $f_Z(x)$, là tỷ lệ của số cạnh có mặt trong cây đúng trong số tất cả các cạnh có giá trị hỗ trợ bootstrap $x\%$ (đếm trên tất cả các cây $T_k^{tốt\ nhất}$) [39]. $f_Z(x)$ phản ánh xác suất một cạnh với giá trị hỗ trợ bootstrap $x\%$ là một cạnh đúng. Phương pháp Z được gọi là không chệch nếu $f_Z(x) = x\%$ cho tất cả các giá trị của x. Khi vẽ đồ thị thể hiện mối liên hệ giữa f_Z và giá trị hỗ trợ bootstrap $x\%$, đồ thị của một phương pháp không chệch sẽ trùng đường chéo nối điểm $(0; 0)$ và $(100; 100)$ (Hình 1.11, đường màu đen). Đường cong của

$f_Z(x)$ nằm phía trên đường chéo có nghĩa phương pháp bootstrap cho ước lượng thấp hơn xác suất đúng của cạnh (tức là phương pháp này bảo thủ) (Hình 1.11, đường màu xanh). Ngược lại đường cong cho $f_Z(x)$ nằm bên dưới đường chéo có nghĩa phương pháp cho ước lượng cao hơn xác suất đúng của cạnh (tức là phương pháp này lạc quan) (Hình 1.11, đường màu đỏ). Lưu ý là trong thực hành, ta chỉ quan tâm đến những cạnh có giá trị hỗ trợ bootstrap $\geq 70\%$, do đó, chỉ phân tích một phương pháp dựa trên nửa bên phải của đồ thị.



Hình 1.11. Ví dụ đồ thị thể hiện độ chuẩn xác của phương pháp bootstrap lạc quan (màu đỏ), phương pháp bảo thủ (màu xanh), phương pháp không chệch (màu đen). Ta chỉ phân tích phần bên phải của đồ thị (x >= 70).

Để sinh sắp hàng gốc mô phỏng cho đánh giá độ chuẩn xác của các phương pháp bootstrap, luận án sử dụng công cụ Seq-Gen [69]. Với dữ liệu vào là cây đúng giả định $T^{đúng}$, một mô hình tiến hóa $M^{đúng}$ và các tham số kích thước cho sắp hàng mong muốn, Seq-Gen sẽ mô phỏng quá trình tiến hóa theo $T^{đúng}$ và $M^{đúng}$ và tạo ra một sắp hàng gốc. Seq-Gen có thể sinh sắp hàng mô phỏng cho cả DNA và protein với nhiều lựa chọn mô hình tiến hóa - bao gồm cả mô hình cho tính không đồng nhất của tốc độ biến đổi giữa các vị trí trên chuỗi.

1.5.4 Các phương pháp hiện tại

Trong việc xác định độ hỗ trợ thống kê cho cây tiến hóa thì làm bootstrap phi tham số là kỹ thuật được công nhận rộng rãi nhất. Do tính toán theo SBS tốn kém nên nhiều nghiên cứu đã đề xuất các phương án dùng heuristic để tăng tốc bootstrap. Các nghiên cứu bootstrap nhanh tập trung cho phân tích theo tiêu chuẩn ML bởi tính toán likehood phức tạp và bởi xây dựng cây tiến hóa theo ML thuộc lớp bài toán NP-khó [10].

Lấy mẫu ước lượng log-likelihood (resampling estimated log-likelihoods – RELL) [36,45] là một trong những nghiên cứu bootstrap nhanh tiên phong. Trong đó, các tác giả sử dụng lại các điểm số log-likelihood của một cây T trên các vị trí của sắp hàng gốc để ước lượng log-likelihood cho T trên một sắp hàng bootstrap (tức điểm RELL của T trên sắp hàng bootstrap). Sau đó, trên 1 tập hợp ví dụ gồm 3 cây cho sắp hàng của 4 loài, các tác giả dùng RELL để tính tần suất mỗi cây là tốt nhất trên các sắp hàng bootstrap. Họ quan sát thấy kết quả thu được xấp xỉ với kết quả tính theo SBS. Trong [4], RELL được vận dụng để tính xác suất bootstrap cục bộ (local bootstrap probabilities – LBP) cho mỗi cạnh trong của cây ML dựa trên việc so sánh 3 cấu trúc NNI ứng viên của cạnh này. Các tác giả dùng thông tin thu được để định hướng thuật toán tìm kiếm. Khác với LBP, phương pháp kiểm định tỉ lệ likelihood xấp xỉ (approximate likelihood-ratio test – aLRT) [5] và phiên bản tổng quát SH-aLRT [33] vận dụng kiểm định SH trên 3 cây NNI này. Các phương pháp LBP và SH-aLRT đều nhanh hơn SBS nhiều nhờ bỏ qua phần tối ưu mô hình tiến hóa và phần tối ưu độ dài cạnh mỗi khi phân tích sắp hàng bootstrap. Độ hỗ trợ chúng gán cho mỗi cạnh chỉ có nghĩa khi so sánh 3 cây NNI ứng viên liên quan và không có ý nghĩa rõ ràng khi 4 cây con gắn với cạnh đang khảo sát bị thay đổi. SH-aLRT được cài đặt trong phần mềm mã nguồn mở PhyML 3.0 [33].

Phương pháp bootstrap nhanh (rapid bootstrap – RBS) [81] của RAxML [79] có khả năng tính độ hỗ trợ thống kê cho mỗi cạnh trong với ý nghĩa sát với phương pháp SBS. RBS duyệt tìm cây tốt nhất theo ML cho mỗi sắp hàng bootstrap bằng một

phiên bản thô sơ của thuật toán Lazy Subtree Rearrangement (LSR) [78] trong đó: bán kính tìm kiếm leo đồi được thu nhỏ, tìm kiếm leo đồi sẽ kết thúc sớm nếu không có khả năng cho điểm số log-likelihood tốt và bước tối ưu tham số mô hình tiến hóa bị lược bỏ. Thực nghiệm trên 22 sắp hàng thực cho thấy phân tích bootstrap bằng RBS nhanh hơn SBS từ 8 tới 20 lần; các giá trị hỗ trợ bootstrap tính bởi RBS có tương quan cao với kết quả SBS. Kết quả phân tích dữ liệu mô phỏng trong [56] cho thấy RBS giống với SBS ở xu hướng ước lượng bootstrap thấp hơn xác suất đúng của cạnh. Bootstrap nhanh được tích hợp vào phần mềm RAxML từ năm 2008 và tiếp tục được duy trì trong phiên bản cải tiến kỹ thuật mới nhất là RAxML-NG [46] giới thiệu năm 2019 có hỗ trợ linh hoạt cho các kiến trúc song song hiệu năng cao.

Các phương pháp nói trên đều dựa trên việc lấy mẫu một phần các vị trí trong sắp hàng gốc (lấy mẫu dữ liệu) để tính độ hỗ trợ thống kê cho cây tiến hóa. Ngoài tiếp cận này, còn có các phương pháp lấy mẫu cây gặp trong quá trình duyệt tìm cây, ví dụ như quartet puzzling [75,82] hay phân tích Bayes [70,99] tuy nhiên chúng đều có chi phí tính toán lớn hơn tiếp cận lấy mẫu dữ liệu [56]. Các phương pháp Bayes (có phần mềm tiêu biểu là MrBayes [70], BEAST [14]) có xu hướng cho độ hỗ trợ thấp hơn xác suất đúng của cạnh nếu có vi phạm mô hình hoặc hiện tượng đa phân [6,13,52,83].

Cần phải nói thêm rằng cách diễn giải độ tin cậy của các phân nhóm trong cây từ giá trị hỗ trợ bootstrap và từ xác suất hậu nghiệm theo Bayes là khác nhau. Việc lựa chọn trong hoàn cảnh nhất định nên dùng độ hỗ trợ nào vẫn là một chủ đề nghiên cứu lớn, có tính triết học và phức tạp [6,81]. Việc thảo luận so sánh các phương pháp luận án đề xuất với các phương pháp Bayes, do đó, không phải trọng tâm của luận án này.

Phương pháp bootstrap siêu nhanh (Ultrafast approximation for phylogenetic bootstrap – UFBoot; chi tiết trình bày trong phần 2.3) [56] là đề xuất mới nhất cho tăng tốc bootstrap. UFBoot tìm ra tập hợp xấp xỉ kết quả của SBS nhờ kết hợp tiếp cận lấy mẫu dữ liệu và tiếp cận lấy mẫu cây. Nó thực hiện duy nhất một lần duyệt tìm

cây trên không gian tìm kiếm gắn với sắp hàng gốc bằng thuật toán IQPNNI [49]. Mỗi cây duyệt trong không gian cây gốc sẽ ngay lập tức được chấm điểm RELL [45] trên từng sắp hàng bootstrap xem nó có phải là cây tốt nhất cho sắp hàng bootstrap đó không. UFBoot nhanh hơn RBS trung bình 3 lần trên DNA và 10 lần trên protein. UFBoot nhanh là do nó tránh được B lần duyệt tìm cây trên B sắp hàng bootstrap và tránh được việc tối ưu các độ dài cạnh và tham số mô hình tiến hóa khi tính log-likelihood của một cây trên một sắp hàng bootstrap.

Ngoài ưu điểm về thời gian chạy, UFBoot cho ước lượng bootstrap sát với xác suất đúng của cạnh nếu được chỉ định mô hình tiến hóa đúng. Việc UFBoot cho kết quả có độ chuẩn xác tốt hơn SBS khẳng định thành công của thuật toán IQPNNI (chi tiết trình bày trong Phần 2.3.2) trong việc khảo sát không gian cây của sắp hàng gốc theo tiêu chuẩn ML. Đây là động lực để chúng tôi thiết kế thuật toán tương tự cho tiêu chuẩn MP trình bày trong Chương 3.

Như đã giới thiệu trong phần 1.3.2, phân tích tiến hóa theo tiêu chuẩn MP có hai phần mềm đa nền tảng tiêu biểu là PAUP [84] và TNT [31] đều hỗ trợ phương pháp bootstrap chuẩn. Phương pháp đề xuất trong Chương 3 sẽ được so sánh với các phương pháp này.

BOOSTER [51], được công bố sau các công trình nghiên cứu liên quan đến luận án này (do đó không được khảo sát trong phần thực nghiệm) là một công thức mới để tính độ hỗ trợ từ tập cây bootstrap chuẩn nhằm cải tiến độ chuẩn xác bootstrap. Nó có mã nguồn mở và cũng được tích hợp vào RAxML-NG [46].

1.6 Kết luận chương

Phân tích cây tiến hóa dựa trên sắp hàng chuỗi phân tử sinh học đóng vai trò quan trọng vì nó làm phong phú thêm hiểu biết của chúng ta về cách gen, bộ gen, các loài (và các chuỗi phân tử sinh học nói chung) tiến hóa. Nhờ phân tích cây tiến hóa, ta không những học được quá trình các chuỗi trở thành như hiện nay, mà còn học được các nguyên lý tổng quát cho phép ta dự đoán chúng sẽ thay đổi như thế nào

trong tương lai. Do vậy, nó có nhiều ứng dụng cả trong và ngoài sinh học. Một số ứng dụng tiêu biểu được tổng hợp trong cuốn sách của Herron và Freeman [38].

Làm bootstrap để xác định độ hỗ trợ thống kê cho cây tiến hóa đang trở nên không thể thiếu trong các công trình nghiên cứu phân tích quan hệ tiến hóa. Khó khăn lớn nhất khi làm bootstrap trong phân tích tiến hóa là chi phí tính toán cao và còn cao hơn nữa với phân tích ML và trên những bộ dữ liệu lớn ra đời nhờ công nghệ giải trình tự thế hệ tiếp theo. Luận án vì thế tập trung nghiên cứu các phương pháp dùng heuristic để cải tiến tốc độ làm bootstrap, đồng thời tìm hiểu giải quyết các vấn đề chung của làm bootstrap theo tiêu chuẩn MP và ML.

Chương 2 PHƯƠNG PHÁP UFBOOT2 GIẢI NHANH BÀI TOÁN XÂY DỰNG CÂY BOOTSTRAP TIẾN HÓA THEO TIÊU CHUẨN HỢP LÝ NHẤT

UFBoot [56] là phương pháp nhanh tìm lời giải chấp nhận được cho bài toán xây dựng cây bootstrap tiến hóa theo tiêu chuẩn hợp lý nhất. Chương này trước tiên khảo sát luồng tính toán trong UFBoot, qua đó, đề xuất phương pháp UFBoot2 hiệu quả hơn về mặt thời gian nhờ tính nhanh likelihood của cây. Ngoài ra, phương pháp UFBoot2 còn tích hợp các giải pháp để giải quyết hiệu quả các vấn đề dữ liệu và mô hình: hiện tượng gán giá trị hỗ trợ bootstrap cao cho các cạnh khi dữ liệu vào không có đủ thông tin tiến hóa, dẫn tới khả năng cao tồn tại đỉnh đa phân (polytomy) hay khi mô hình có vi phạm giả thiết nghiêm trọng; và cải tiến mở rộng để có thể phân tích sắp hàng nhiều gen.

2.1 Giới thiệu về xây dựng cây tiến hóa theo tiêu chuẩn hợp lý nhất

Khái niệm độ hợp lý (sau đây gọi là likehood) thường dùng trong khoa học tự nhiên khi được cho dữ liệu D và cần phải ra quyết định về một giải thích hợp lý về dữ liệu.

Trong phân tích bootstrap tiến hóa theo tiêu chuẩn hợp lý nhất (maximum likelihood – ML), việc xây dựng cây tốt nhất cho sắp hàng gốc và việc xây dựng tập cây bootstrap đều sử dụng tiêu chuẩn ML. Tức là, mỗi mũi tên đậm màu xanh trong lược đồ minh họa ở Hình 1.9 tương đương với một lần xây dựng cây tiến hóa theo tiêu chuẩn ML. Phương pháp bootstrap được minh họa trong Hình 1.9 khi ấy gọi là bootstrap chuẩn (SBS) theo ML.

Phần sau đây sẽ trình bày về xây dựng cây tiến hóa theo tiêu chuẩn ML cho $A = (X_1, \dots, X_n)$ là một sắp hàng gồm n chuỗi với độ dài m và sử dụng mô hình tiến hóa (hay ma trận tốc độ biến đổi tức thì) Q. Chúng ta cần xây dựng một cây tiến hóa T có

thể giải thích một cách hợp lý nhất quá trình biến đổi thành các chuỗi trong A theo cây T và mô hình biến đổi Q. Hay nói cách khác, $H = (T, Q)$ là một giả thuyết giải thích quá trình biến đổi thành các chuỗi X_j theo cây T và mô hình Q. Chúng ta cần tìm giả thuyết có likelihood (tính hợp lý) cao nhất.

Để xây dựng cây T giải thích hợp lý nhất sắp hàng A dựa vào mô hình tiến hóa Q, chúng ta có một số giả thiết sau:

- Quá trình tiến hóa giữa các vị trí trên sắp hàng là độc lập với nhau.
- Quá trình tiến hóa của tất cả các vị trí trên sắp hàng đều tuân theo mô hình Q và cây T.

Likelihood $L(T, Q|A)$ của cây T và mô hình tiến hóa Q để giải thích sắp hàng A được tính theo công thức sau:

$$L(T, Q|A) = P(A|T, Q)$$

trong đó $P(A|T, Q)$ là xác suất để quan sát được dữ liệu A với điều kiện cây T và mô hình biến đổi Q là đúng.

Do giả thiết rằng quá trình tiến hóa ở các vị trí trên sắp hàng A là độc lập và giống nhau theo cây T, likelihood của cây T có thể được tính chi tiết như sau:

$$L(T, Q|A) = P(A|T, Q) = \prod_{i=1}^{m} P(A_i|T, Q) \quad (2.1)$$

trong đó A_i là cột thứ i trên sắp hàng A. Ta thấy, $L(T, Q|A_i) = P(A_i|T, Q)$ là likelihood của cây T khi giải thích dữ liệu tại ví trí thứ i trên sắp hàng A theo mô hình tiến hóa Q.

Để không bị mất mát thông tin khi tính toán trên các số bé, thay vì tính likelihood, người ta thường tính log-likelihood của cây. Ký hiệu log-likelihood của cây tính trên cả sắp hàng là $\ell(T, Q|A)$, tính trên vị trí i của sắp hàng là $\ell(T, Q|A_i)$. Ta có:

$$\ell(T, Q|A) = \sum_{i=1}^{m} \ell(T, Q|A_i) \quad (2.2)$$

Như đã giới thiệu trong phần 1.3.4, xây dựng cây tiến hóa theo tiêu chuẩn ML thực chất gồm hai bước tối ưu: (i) với một cấu trúc phân nhánh T cụ thể, tối ưu các tham số của mô hình tiến hóa Q và tối ưu độ dài các cạnh để tính điểm $\ell(T, Q|A)$ và (ii) khám phá không gian các cấu trúc phân nhánh ứng viên để tìm cấu trúc phân nhánh T^* làm cực đại hàm log-likelihood.

$$T^* = argmax_T \{\ell(T, Q|A)\}$$

Đây là một bài toán khó và phức tạp, nó thuộc lớp các bài toán NP-khó [10].

2.2 Thuật toán pruning để tính likelihood cây

Phần này tóm tắt lại phương pháp tính likelihood trên một mô hình tiến hóa có các tham số đã xác định cho một cây đã biết độ dài cạnh theo hai cách: (i) sử dụng định nghĩa và (ii) sử dụng thuật toán pruning như được trình bày trong [22] để thấy được vai trò quan trọng của thuật toán pruning trong xây dựng cây tiến hóa theo tiêu chuẩn ML. Ví dụ đưa ra minh họa việc tính toán cho các chuỗi DNA nhưng có thể tổng quát hóa cho tất cả các mô hình ký tự rời rạc.

2.2.1 Tính likelihood cho một cây theo định nghĩa

Giả sử ta có một sắp hàng DNA gồm m vị trí. Ta được cho một cây biết độ dài cạnh và một mô hình tiến hóa Q cho phép tính các xác suất biến đổi trạng thái trên cây này. Cụ thể, mô hình cho phép tính các xác suất chuyển trạng thái $p_{ij}(t)$, là xác suất trạng thái j sẽ tồn tại ở điểm kết thúc của một cạnh độ dài t, nếu trạng thái ở điểm bắt đầu của cạnh là i. Lưu ý: để tiện lợi t được gọi là "thời gian" trong luận án này nhưng nó là độ dài cạnh chứ không phải thời gian thực sự. Để tính likelihood, ta cần sử dụng 2 giả thiết:

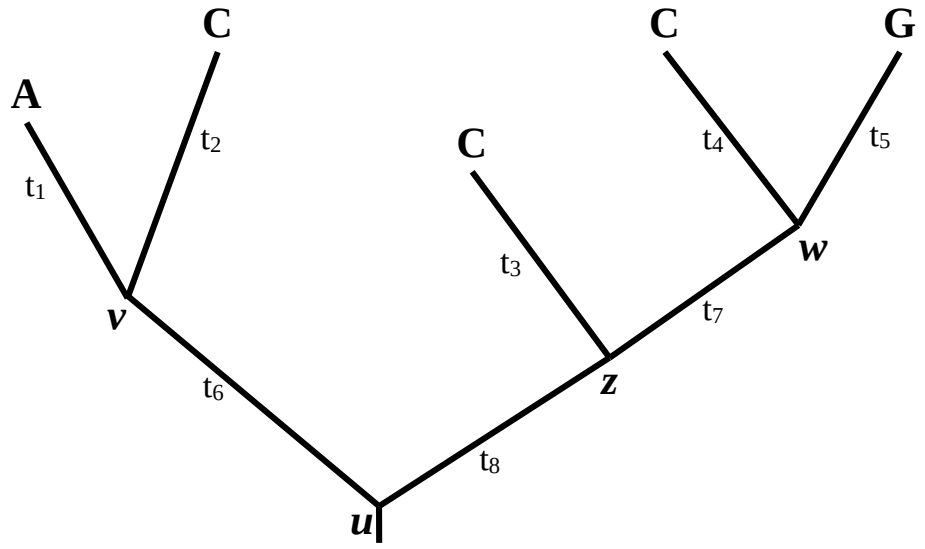
1. Việc tiến hóa của các vị trí khác nhau (trên cùng 1 cây) là độc lập.

2. Việc tiến hóa của các cạnh khác nhau là độc lập.

Giả thiết đầu cho phép phân rã likelihood thành một tích, mỗi nhân tử ứng với một vị trí trên sắp hàng như trong (2.1). Suy ra, để tính likelihood cho cây trên một sắp hàng ta chỉ cần biết cách tính likelihood cho cây tại một vị trí đơn lẻ. Với ví dụ trong Hình 2.1, likelihood cho cây tại ví trí minh họa A_i là một tổng, trên tất cả các nucleotide ứng viên có thể tồn tại ở các đỉnh trong của cây, của các xác suất cho từng ngữ cảnh biến cố:

$$P(A_i|T) = \sum_u \sum_v \sum_z \sum_w P(\mathrm{A}, \mathrm{C}, \mathrm{C}, \mathrm{C}, \mathrm{G}, u, v, z, w|T) \tag{2.3}$$

mỗi tổng chạy trên tất cả bốn nucleotide.



Hình 2.1. Một cây biết độ dài cạnh và dữ liệu tại một vị trí đơn lẻ trên sắp hàng. Ví dụ này để minh họa tính likelihood bằng định nghĩa và bằng thuật toán pruning. Đỉnh gốc là *u*.

Giả thiết về tính độc lập của việc tiến hóa tại từng cạnh cho phép phân rã xác suất ở vế phải của (2.3) thành một tích của các nhân tử:

$$\begin{aligned} P(\mathrm{A}, \mathrm{C}, \mathrm{C}, \mathrm{C}, \mathrm{G}, u, v, z, w|T) = P(u)\, & p_{uv}(t_6)\, p_{v\mathrm{A}}(t_1)\, p_{v\mathrm{C}}(t_2) \\ & p_{uz}(t_8)\, p_{z\mathrm{C}}(t_3)\, p_{zw}(t_7) \\ & p_{w\mathrm{C}}(t_4)\, p_{w\mathrm{G}}(t_5) \end{aligned} \tag{2.4}$$

Trong thực hành, ta đặt $P(u)$ bằng xác suất cân bằng của ký tự trạng thái u theo mô hình biến đổi nucleotide. Những xác suất khác được tính từ mô hình biến đổi nucleotide theo công thức (1.2). Biến đổi ở mỗi cạnh độc lập với tất cả các cạnh khác nếu ký tự trạng thái ở điểm bắt đầu cạnh ấy đã xác định.

Biểu thức (2.4) khó tính toán do có nhiều hạng tử bên trong. Với mỗi vị trí sắp hàng, ta tính tổng của $4^4 = 256$ hạng tử. Số hạng tử tăng theo hàm mũ so với số lượng loài. Trên một cây có gốc có n loài, ta có $n-1$ đỉnh trong, mỗi đỉnh trong có thể nhận một trong 4 trạng thái. Do đó ta cần 4^{n-1} hạng tử.

2.2.2 Tính likelihood cho một cây theo thuật toán pruning

Thuật toán pruning [20] tính toán hiệu quả likelihood của cây nhờ sử dụng tiếp cận quy hoạch động. Ý tưởng là đẩy các ký hiệu tổng trong (2.3) đi càng xa càng tốt và nhóm nó trong cặp ngoặc khi có thể.

$$P(A_i|T) = \sum_u \sum_v \sum_z \sum_w P(u)\, p_{uv}(t_6)\, p_{v\mathrm{A}}(t_1)\, p_{v\mathrm{C}}(t_2)\, p_{uz}(t_8)\, p_{z\mathrm{C}}(t_3)\, p_{zw}(t_7)\, p_{w\mathrm{C}}(t_4)\, p_{w\mathrm{G}}(t_5) \tag{2.5}$$

tương đương

$$P(A_i|T) = \sum_u P(u) \left(\sum_v p_{uv}(t_6)\, p_{v\mathrm{A}}(t_1)\, p_{v\mathrm{C}}(t_2) \right) \times \left(\sum_z p_{uz}(t_8)\, p_{z\mathrm{C}}(t_3) \times \left(\sum_w p_{zw}(t_7)\, p_{w\mathrm{C}}(t_4)\, p_{w\mathrm{G}}(t_5) \right) \right) \tag{2.6}$$

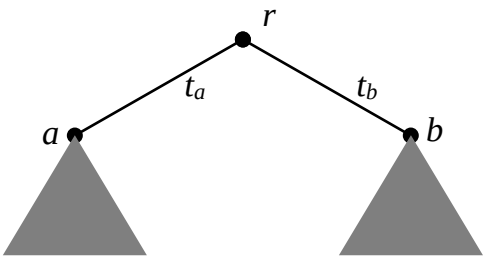
Việc tính toán theo (2.6) diễn ra từ cặp ngoặc sâu nhất ra ngoài. Điều này gợi ý luồng tính toán trong cây là hướng xuống gốc.

Thuật toán pruning dựa trên hàm likelihood riêng phần của một cây con, ký hiệu là $L_i^r(x)$ với $x \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$. Đây là xác suất quan sát được mọi thứ từ đỉnh r trở đến các lá, tại vị trí sắp hàng thứ i, với điều kiện là đỉnh r có trạng thái x. Trong (2.6), hạng tử

$$p_{w\mathrm{C}}(t_4)\ p_{w\mathrm{G}}(t_5)$$

là một trong những đại lượng này. Có 4 đại lượng như vậy ứng với các trạng thái khác nhau tại đỉnh w. Điểm mấu chốt của thuật toán pruning là, một khi 4 con số này đã được tính thì ta không cần liên tục tính lại chúng.

Thuật toán pruning tiến hành nhờ tính toán đệ quy hàm $L_i^r(x)$ tại mỗi đỉnh trên cây theo chính hàm này này tại các đỉnh hậu duệ gần nhất. Giả sử đỉnh r có 2 hậu duệ gần nhất là a và b, là các đỉnh kết thúc của các cạnh có độ dài t_a và t_b (như minh họa trong Hình 2.2).



Hình 2.2. Một cây *T* để minh họa thuật toán pruning và pruning nhanh. Nó được định gốc ngẫu nhiên tại điểm *r* trên cạnh (*a,b).* Gốc cách 2 đầu cạnh khoảng tương ứng là t_a và t_b.

Ta có

$$L_i^r(x) = \left(\sum_s p_{xs}(t_a) L_i^a(s)\right)\left(\sum_y p_{xy}(t_b) L_i^b(y)\right) \tag{2.7}$$

Trước khi bắt đầu tính toán, ta khởi tạo likelihood riêng phần ở các đỉnh lá như sau: nếu đỉnh lá h có trạng thái A thì các giá trị L_i^h của đỉnh lá đó sẽ là

$$\left(L_i^h(\mathrm{A}), L_i^h(\mathrm{C}), L_i^h(\mathrm{G}), L_i^h(\mathrm{T})\right) = (1,0,0,0) \tag{2.8}$$

Thuật toán bắt đầu từ một đỉnh có tất cả đỉnh hậu duệ gần nhất đều là lá (trong cây luôn có ít nhất một đỉnh trong như vậy). Sau đó nó lần lượt tính cho các đỉnh gần gốc hơn, chỉ áp dụng với các đỉnh có tất cả hậu duệ gần nhất đã được tính likelihood. Kết quả là L_i^r cho đỉnh gốc. Ta hoàn thiện việc tính likelihood cho vị trí sắp hàng này bằng cách tính trung bình có trọng số trên tất cả 4 ký tự trạng thái, sử dụng trọng số là xác suất tiền nghiệm theo mô hình xác suất:

$$L_i = \sum_x \pi_x L_i^r(x) \tag{2.9}$$

Theo nguyên lý Pulley [20], khi Q thuận nghịch, likelihood của một vị trí sắp hàng *i* nào đó không thay đổi miễn là $t_a + t_b = t$ không đổi. Nói cách khác, ta có thể đặt $t_a = 0$ (di chuyển *r* tới *a*). Khi đó, $P(t_a) = P(0)$ trở thành ma trận đơn vị và kết hợp (2.7) và (2.9) cho ta:

$$L_i = \sum_x \pi_x L_i^a(x) \left(\sum_y p_{xy}(t) L_i^b(y) \right) \tag{2.10}$$

Về cơ bản, thuật toán của Felsenstein là thuật toán quy hoạch động. Nó có độ phức tạp thời gian là $O(nmc^2)$, trong đó *n*, *m* và *c* lần lượt là số chuỗi, số vị trí sắp hàng và số ký tự trạng thái (dữ liệu DNA thì *c*=4). Độ phức tạp không gian là $O(nmc)$ nhằm lưu các vector likelihood riêng phần cho tất cả các đỉnh trong của cây.

Như vậy, thuật toán pruning cho phép tính nhanh likelihood của cây. Với các cây không gốc và với mô hình tiến hóa có tính thuận nghịch thời gian thì likelihood không thay đổi khi ta định gốc ở các cạnh khác nhau trong cây. Khi định gốc ở giữa một cạnh nào đó, ta có thể áp dụng công thức đệ quy về 2 đầu của cạnh và tính likelihood của cây rất nhanh với một độ dài cụ thể của cạnh này. Điều đó có nghĩa là ta có thể nhanh chóng tìm likelihood cực đại khi cho biến thiên độ dài của một cạnh trong một cấu trúc cây cho trước (và cố định độ dài tất cả các cạnh khác). Khi ta luân phiên làm việc này với các cạnh khác nhau trong cây, ta sẽ nhanh chóng tìm được độ dài tối ưu của các cạnh (ứng với cấu trúc cây đó). Với bài toán xây dựng cây tiến hóa theo tiêu chuẩn ML, thuật toán pruning cho phép đánh giá một cách hiệu quả các biến đổi cấu trúc cục bộ.

2.3 Thuật toán UFBoot

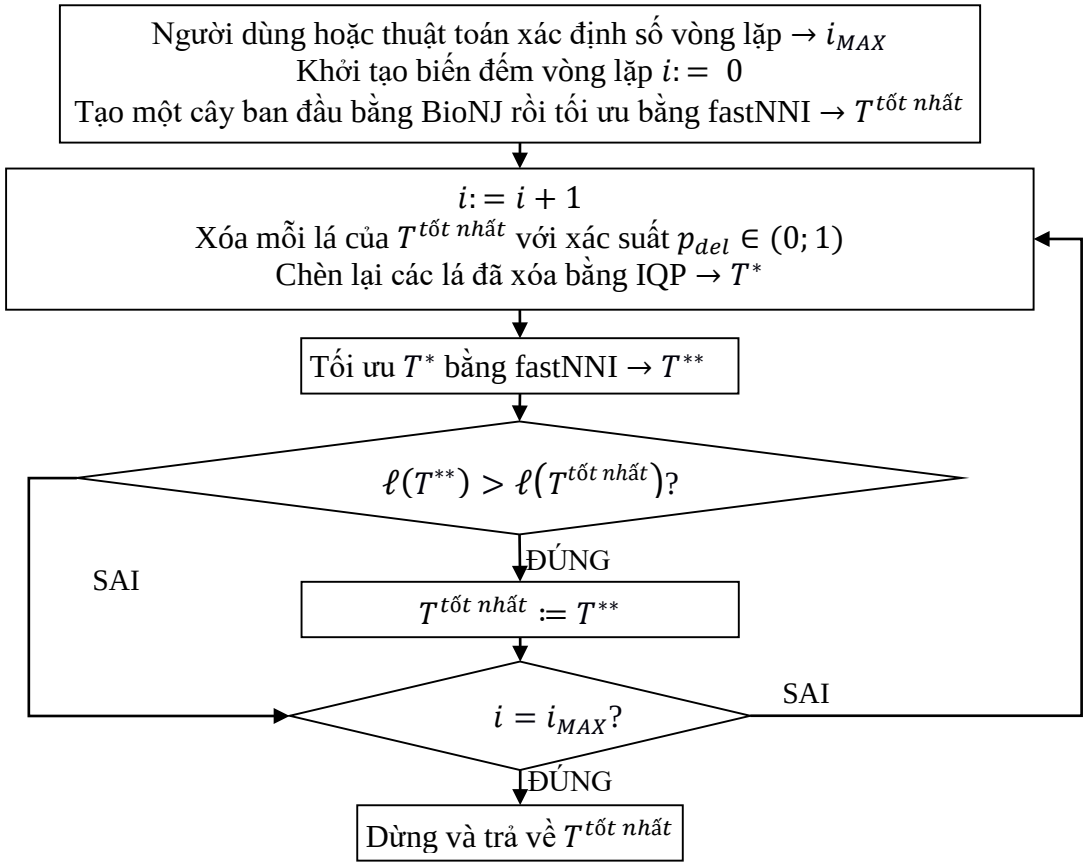
2.3.1 Tóm tắt ý tưởng

Ý tưởng chính của UFBoot (**Thuật toán 2.1**) là giữ lại các cây duyệt khi thực hiện tìm kiếm trên sắp hàng gốc và tính likelihood cho chúng trên các sắp hàng bootstrap. Để tăng tốc độ tính toán likelihood hơn nữa đối với các sắp hàng bootstrap, UFBoot sử dụng chiến lược lấy mẫu ước lượng log-likelihood (RELL) [45]. Đối với mỗi sắp hàng bootstrap, cây có điểm RELL cao nhất biểu thị cây bootstrap theo ML. Trái ngược với SBS, UFBoot không tối ưu cây này. Sự khác biệt giữa UFBoot và SBS trong giá trị hỗ trợ bootstrap gán cho các cạnh là do UFBoot và SBS chọn cây bootstrap theo cách khác nhau.

2.3.2 Thuật toán IQPNNI

Trong thuật toán UFBoot, việc lấy mẫu cây trên sắp hàng gốc được thực hiện nhờ thuật toán IQPNNI [49]. IQPNNI (tóm tắt trong Hình 2.3) sử dụng thuật toán BioNJ [27] ghép hàng xóm dựa vào khoảng cách để xây dựng cây khởi tạo và dùng thuật toán fastNNI [34] để tối ưu likelihood và cập nhật cây tốt nhất $T^{tốt\ nhất}$.

IQPNNI dùng một chiến lược thông minh để thoát khỏi cực trị địa phương. Khi fastNNI không tìm được likelihood tốt hơn, IQPNNI tạo ra cây mới T^* bằng cách xóa ngẫu nhiên một số loài khỏi cây $T^{tốt\ nhất}$, sau đó chèn chúng vào cây bằng IQP - một thuật toán nhanh khai thác các cấu trúc phân nhánh không gốc 4 lá (quartet) tính từ dữ liệu. Thuật toán tiếp tục tối ưu T^* bằng fastNNI để thu được cây T^{**}. Cây $T^{tốt\ nhất}$ được cập nhật bằng cây T^{**} nếu cây T^{**} có likelihood tốt hơn. Thủ tục IQP theo sau là fastNNI được lặp lại theo số vòng lặp do người dùng phần mềm chỉ định hoặc theo điều kiện dừng mà thuật toán cài sẵn dựa trên thống kê.



Hình 2.3. Sơ đồ khối thuật toán IQPNNI.

2.3.3 Công thức RELL

Ký hiệu A^{data} là sắp hàng gốc của *n* chuỗi và *m* vị trí; các vị trí này được nhóm thành các mẫu-vị trí $D_1, D_2, \dots, D_k$ với tần suất tương ứng là $d_1, d_2, \dots, d_k$. Điểm log-likelihood cho cấu trúc cây *T* khi biết A^{data} được tính bởi công thức:

$$\ell(T|A^{data}) = \sum_{i=1}^{k} \ell(T|D_i) \times d_i \tag{2.11}$$

trong đó $\ell(T|D_i)$ là điểm log-likelihood cho cây *T* tại mẫu-vị trí D_i, được tính toán hiệu quả với thuật toán pruning [19,20]. Thuật toán này đã được trình bày kĩ ở phần 2.2.

Với một cây T đã được tính điểm log-likelihood tại các mẫu-vị trí của sắp hàng gốc, chiến lược RELL cho phép ta tính xấp xỉ điểm log-likelihood của T trên sắp hàng bootstrap $A^{bootstrap}$ cực nhanh chỉ với phép tính tổng:

$$\ell(T|A^{bootstrap}) \approx \hat{\ell}(T|A^{bootstrap}) = \sum_{i=1}^{k} \ell(T|D_i) \times d_i^{bootstrap} \quad (2.12)$$

Trong đó $d_i^{bootstrap}$ là tần suất của D_i trong $A^{bootstrap}$. Nhờ vậy, ta tránh được việc gọi tới thuật toán pruning khi tính log-likelihood cho T tại mỗi mẫu-vị trí của $A^{bootstrap}$.

2.3.4 Giả mã của thuật toán UFBoot

Thuật toán 2.1. Thuật toán UFBoot

Dữ liệu vào: Sắp hàng gốc A^{data} gồm n chuỗi (taxa). Mô hình tiến hóa Q cho phép tính các xác suất biến đổi trạng thái trên cây.

Dữ liệu ra: Tập cây bootstrap. Cây ML xây dựng cho A^{data} được gắn các giá trị hỗ trợ bootstrap cho mỗi cạnh.

Bắt đầu

1) Bước khởi đầu: Tạo B sắp hàng bootstrap, $A_1, A_2, \dots, A_B$. Với mỗi sắp hàng bootstrap A_b khởi tạo cây bootstrap $T_b := null$ và điểm RELL bootstrap $\hat{\ell}(T_b|A_b) := -\infty$. Khởi tạo một tập cây $S := \{\}$ và ngưỡng log-likelihood $\ell_{min} := -\infty$.
2) Bước khám phá: Tiến hành tìm kiếm cây sử dụng thuật toán IQPNNI [49] cho sắp hàng gốc A^{data} . Mỗi khi duyệt một cây mới $T \notin S$ thỏa mãn $\ell(T|A^{data}) \geq \ell_{min}$, thêm T vào S và tính $\hat{\ell}(T|A_b)$, cho các $b = 1, \dots, B$ dựa trên (2.12). Nếu $\hat{\ell}(T|A_b) > \hat{\ell}(T_b|A_b)$, cập nhật $T_b := T$.
   Khi hết mỗi lượt lặp tìm kiếm của thuật toán IQPNNI, cập nhật ℓ_{min}.
3) Bước tóm tắt: Xây dựng một cây đồng thuận từ các cây bootstrap $\{T_1, T_2, \dots, T_B\}$, hoặc tính và gắn giá trị hỗ trợ bootstrap lên cây ML mà IQPNNI tìm được cho A^{data}.

Kết thúc

2.3.5 Thuật toán pruning ước lượng độ dài cạnh

Để tính log-likelihood $\ell(T|A^{data})$ cho một cấu trúc phân nhánh T cụ thể trên sắp hàng gốc A^{data}, UFBoot duyệt các đỉnh trong T để tối ưu mỗi lần một cạnh bằng phương pháp Newton-Raphson [63]. Duyệt cây được lặp lại tới khi log-likelihood hội tụ. Trong quá trình này, thao tác phổ biến là tính $\ell(T|A^{data})$ khi biết cạnh (a, b) nối đỉnh a và b có độ dài *t*. Bởi (2.10) được áp dụng lặp lại khi tối ưu *t*, ta cần tính sẵn các vector likelihood riêng phần $L_i^a(.)$ và $L_i^b(.)$ để tiết kiệm tính toán. **Thuật toán 2.2** tóm tắt việc tối ưu t nhờ tính sẵn các vector L theo thuật toán pruning. Chi phí tính toán cho (2.10) là mc^2 với một độ dài *t* cho trước.

Thuật toán 2.2. Thuật toán pruning ước lượng độ dài cạnh

Dữ liệu vào: Sắp hàng gốc A^{data} gồm n chuỗi (taxa); mô hình tiến hóa Q; cây T có độ dài cạnh; cạnh (a,b) cần tối ưu độ dài cạnh

Dữ liệu ra: Cây T có cạnh (a,b) đã được tối ưu và log-likelihood tương ứng cho cây

Bắt đầu

1) Thực hiện duyệt các đỉnh trong cây theo thứ tự sau để tính các vector L cho tất cả các đỉnh dựa trên các vector L của hậu duệ sử dụng (2.7).
2) Lặp với từng giá trị của t theo Newton-Raphson tới khi log-likelihood hội tụ:
    - Vận dụng (2.10) để tính $\ell(T|A^{data})$ cho t mới biết rằng L_i^a và L_i^b đã được tính trước đó.

Kết thúc

2.4 Đề xuất thuật toán UFBoot2

2.4.1 Cải tiến tốc độ

Chúng tôi đề xuất một phiên bản hiệu quả hơn về mặt tính toán cho thuật toán pruning của Felsenstein [20]. Giả sử mô hình tiến hóa của chuỗi là mô hình Markov

thuận nghịch. Không mất tính tổng quát, dưới đây chúng tôi sẽ trình bày phương pháp đề xuất này cho các chuỗi DNA với các ký tự trạng thái là $\{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ và các tốc độ là độc lập và cùng phân bố.

2.4.1.1 Phân tích giá trị đặc trưng

Trước khi giải thích về phiên bản đề xuất cho việc cải thiện tốc độ tính toán chiều dài cạnh, luận án sẽ bắt đầu với một đặc điểm cụ thể của ma trận tốc độ thuận nghịch và dạng biến đổi tương tự của nó.

Gọi Q là ma trận tốc độ của mô hình thuận nghịch thời gian tổng quát [48] và $\Pi = \mathrm{diag}(\pi_\mathrm{A}, \pi_\mathrm{C}, \pi_\mathrm{G}, \pi_\mathrm{T})$ là ma trận chéo của tần suất trạng thái cân bằng. Ta có, ma trận

$$Q_1 = \Pi^{1/2} \cdot Q \cdot \Pi^{-1/2}$$

là đối xứng với các giá trị đặc trưng thực và các vector đặc trưng thực. Ngoài ra, ta có thể tính một ma trận trực giao W từ các vector đặc trưng của Q_1 sao cho

$\Lambda = W^T \cdot Q_1 \cdot W$, với $W^T \cdot W = I = \mathrm{diag}(1,1,1,1)$.

Λ là ma trận chéo với các giá trị đặc trưng của Q_1 (và cũng là của Q).

Ta thu được

$$\Lambda = W^T \cdot \Pi^{1/2} \cdot Q \cdot \Pi^{-1/2} \cdot W$$

Do tính kết hợp của phép nhân ma trận

$$W^T \cdot \Pi^{1/2} \cdot \Pi^{-1/2} \cdot W = I.$$

Do đó, $U^{-1} = W^T \cdot \Pi^{1/2}$ và $U = \Pi^{-1/2} \cdot W$; với U là ma trận của các vector đặc trưng của Q.

$u_{xy}^{-1} = w_{xy}^T \sqrt{\pi_y}$ và $u_{yx} = w_{yx}/\sqrt{\pi_y}$. Vì $w_{xy}^T = w_{yx}$ ta thu được

$$u_{xy}^{-1} = \pi_y u_{yx} \tag{2.13}$$

(2.13) sẽ được dùng về sau.

2.4.1.2 Tăng tốc ước lượng độ dài cạnh

Như đã chỉ ra trong phần 2.3.4, chi phí tính toán cho (2.10) là mc^2 với một độ dài *t* cho trước. Ở phần sau đây, luận án trình bày kỹ thuật để giảm chi phí này thành mc, tức là nhanh hơn c lần phiên bản thuần túy của (2.10).

Như đã biến đổi ở phần trước:

$$Q = U \cdot \Lambda \cdot U^{-1}.$$

Do đó,

$$P(t) = e^{Qt} = U \cdot e^{\Lambda t} \cdot U^{-1}.$$

$e^{\Lambda t}$ là ma trận chéo của các hàm mũ của giá trị đặc trưng. Nói cách khác, ta có:

$$p_{xy}(t) = \sum_{z} u_{xz} e^{\lambda_z t} u_{zy}^{-1} \tag{2.14}$$

với mọi trạng thái x và y, trong đó u_{xz} và u_{zy}^{-1} là các phần tử của ma trận các vector đặc trưng U và U^{-1}. Thay vế phải của (2.14) vào (2.10) ta được

$$L_i = \sum_{x} \pi_x L_i^a(x) \left(\sum_{y} \sum_{z} u_{xz} e^{\lambda_z t} u_{zy}^{-1} L_i^b(y) \right).$$

Sắp xếp lại các thành phần trong công thức để có thể rút gọn bằng $\pi_x u_{xz} = u_{zx}^{-1}$ theo (2.13), ta được:

$$L_i = \sum_{z} e^{\lambda_z t} \left(\sum_{x} u_{zx}^{-1} L_i^a(x) \right) \left(\sum_{y} u_{zy}^{-1} L_i^b(y) \right) \tag{2.15}$$

Ký hiệu 2 tổng trong ngoặc tròn của (2.15) là $V_i^a(z)$ và $V_i^b(z)$, ta có:

$$L_i = \sum_z e^{\lambda_z t} V_i^a(z) V_i^b(z) \tag{2.16}$$

So sánh (2.10) và (2.16), ta thấy đã rút gọn từ 2 phép tổng lồng nhau thành chỉ 1 phép tổng, nhờ việc lưu trữ 2 vector $V_i^a(z)$ và $V_i^b(z)$ thay cho các vector likelihood riêng phần $L_i^a(x)$ và $L_i^b(x)$. Chi phí tính toán các vector V tốn gấp đôi chi phí cho các vector likelihood riêng phần, nhưng bù lại, việc ước lượng độ dài cạnh sử dụng (2.16) nhanh hơn *c* lần sử dụng (2.10).

2.4.1.3 Đề xuất thuật toán pruning nhanh

Thuật toán pruning mới sẽ tính và lưu V thay cho việc lưu vector likelihood riêng phần L cho từng đỉnh trong của cây. Để làm việc này, thay vế phải của (2.14) vào (2.7) cho ta:

$$L_i^r(x) = \left(\sum_y \sum_z u_{xz} e^{\lambda_z t_a} u_{zy}^{-1} L_i^a(y) \right) \left(\sum_y \sum_z u_{xz} e^{\lambda_z t_b} u_{zy}^{-1} L_i^b(y) \right).$$

Sắp xếp lại các thành phần trong công thức và thay *L* bằng *V*:

$$L_i^r(x) = \left(\sum_z u_{xz} e^{\lambda_z t_a} V_i^a(z) \right) \left(\sum_z u_{xz} e^{\lambda_z t_b} V_i^b(z) \right) \tag{2.17}$$

Vector *V* của gốc được tính bởi công thức:

$$V_i^r(z) = \sum_x u_{zx}^{-1} L_i^r(x) \tag{2.18}$$

Luận án đề xuất thuật toán pruning nhanh (xem **Thuật toán 2.3**) nhờ kết hợp các thành phần nói trên. Từ đây, luận án đề xuất thuật toán UFBoot2 để làm bootstrap nhanh theo tiêu chuẩn ML bằng việc thay thế toàn bộ thuật toán pruning trong UFBoot (**Thuật toán 2.1**) bằng thuật toán pruning nhanh. UFBoot2 đã được chúng

tôi cài đặt thành công trong hệ thống IQ-TREE (mã nguồn mở cung cấp tại http://www.iqtree.org).

Thuật toán 2.3. Thuật toán pruning nhanh ước lượng độ dài cạnh

Dữ liệu vào: Sắp hàng gốc A^{data} gồm n chuỗi (taxa); mô hình tiến hóa Q; cây T có độ dài cạnh; cạnh (a,b) cần tối ưu độ dài cạnh

Dữ liệu ra: Cây T có cạnh (a,b) đã được tối ưu và log-likelihood tương ứng cho cây

Bắt đầu

1) Thực hiện duyệt các đỉnh trong cây theo thứ tự sau để tính các vector V cho tất cả các đỉnh dựa trên các vector V của hậu duệ sử dụng (2.17) và (2.18).
2) Lặp với từng giá trị của t theo Newton-Raphson tới khi log-likelihood hội tụ:
    - Vận dụng (2.16) để tính $\ell(T|A^{data})$ cho t mới biết rằng V_i^a và V_i^b đã được tính trước đó.

Kết thúc

So sánh thuật toán pruning nhanh (**Thuật toán 2.3**) và thuật toán pruning (**Thuật toán 2.2**) trong ước lượng độ dài cạnh, ta thấy bước 1 của thuật toán pruning nhanh tốn kém hơn, tuy nhiên bước 2 của nó có chi phí tính toán thấp hơn. Cụ thể như sau: Với c là ký hiệu số ký tự trạng thái, ta có $c = 4,20,61$ tương ứng cho mô hình DNA, protein và codon. Bước 1 theo pruning dùng (2.7) tốn $O(c)$ phép nhân để tính $L_i^r(x)$; theo pruning nhanh tốn $O(c)$ phép nhân để dùng (2.17) tính $L_i^r(x)$ từ hậu duệ của r cộng với $O(c)$ phép nhân để dùng (2.18) tính $V_i^r(z)$ từ $L_i^r(x)$. Do vậy, bước 1 theo pruning nhanh tốn gấp đôi chi phí tính toán. Ở bước 2, do đã tính sẵn các vector L_i (cho một vị trí sắp hàng) nên khi thay đổi t, thuật toán pruning tính lại likelihood L_i bằng (2.10) cần 2 vòng lặp lồng nhau tốn $O(c^2)$ phép nhân; trong khi pruning nhanh dùng (2.16) chỉ cần 1 vòng lặp tốn $O(c)$ phép nhân.

Tóm lại, so với pruning, bước 1 của pruning nhanh tốn chi phí tính toán gấp đôi nhưng bước 2 nhanh hơn c lần. Bước 2 chiếm chủ yếu thời gian tối ưu t do được thực hiện lặp lại nhiều lần trên các giá trị khác nhau cho t theo phương pháp Newton-Raphson [63]. Do đó, thuật toán pruning nhanh sẽ giúp tiết kiệm đáng kể chi phí tính $\ell(T|A^{data})$.

2.4.1.4 Cải tiến tốc độ bằng kỹ thuật tối ưu mã nguồn

Ngoài cải tiến về thuật toán, luận án đề xuất khai thác khả năng tính toán đơn lệnh đa dữ liệu (single instruction, multiple data – SIMD), còn gọi là tính toán vector hay tính toán song song dữ liệu, của máy tính hiện đại để tính đồng thời log-likelihood cho k vị trí sắp hàng tùy theo tập lệnh mở rộng mà bộ vi xử lý hỗ trợ.

Theo tổng hợp trong [44], tính toán vector đã được triển khai cách đây hơn 50 năm nhưng chỉ trên các siêu máy tính do đòi hỏi nhiều tài nguyên phần cứng. Nhờ tiến bộ công nghệ, nó bắt đầu được triển khai trên máy tính phổ thông từ giữa thập niên 1990 và hiện được hỗ trợ bởi hầu hết các bộ vi xử lý tuy có giới hạn. Ý tưởng cơ bản là cho phép thực hiện các phép toán không chỉ trên các cặp biến số đơn lẻ mà còn trên các cặp mảng 1 chiều, các cặp ma trận hay các cặp dữ liệu nhiều chiều. Các tập lệnh mở rộng SIMD cho kiến trúc tập lệnh x86 của Intel là công nghệ vi xử lý hỗ trợ tính toán vector tiên tiến nhất hiện nay. Trong đó, ra đời đầu tiên là tập lệnh MMX cho phép tính toán vector sử dụng thanh ghi 64 bit; sau đó là tập lệnh SSE (streaming SIMD extensions) mở rộng cho thanh ghi 128 bit; rồi đến tập lệnh AVX (advanced vector extensions), AVX2 mở rộng cho thanh ghi 256 bit; AVX-512 hiện là tập lệnh mới nhất cho phép tính toán vector sử dụng thanh ghi 512 bit.

SIMD là công nghệ hứa hẹn rút gọn đáng kể chi phí tính toán. Xét một ví dụ trên kiểu số nguyên 32 bit: ta muốn cộng mảng a gồm 8 phần tử với mảng b gồm 8 phần tử và lưu kết quả vào mảng c gồm 8 phần tử. Nếu không sử dụng tính toán vector, ta mất 8 lần lặp lại c[i]=a[i]+b[i] (i là biến điều khiển vòng lặp). Nếu có sử dụng tính toán vector, chẳng hạn trên bộ vi xử lý có tập lệnh AVX2, ta chỉ mất 1 lần

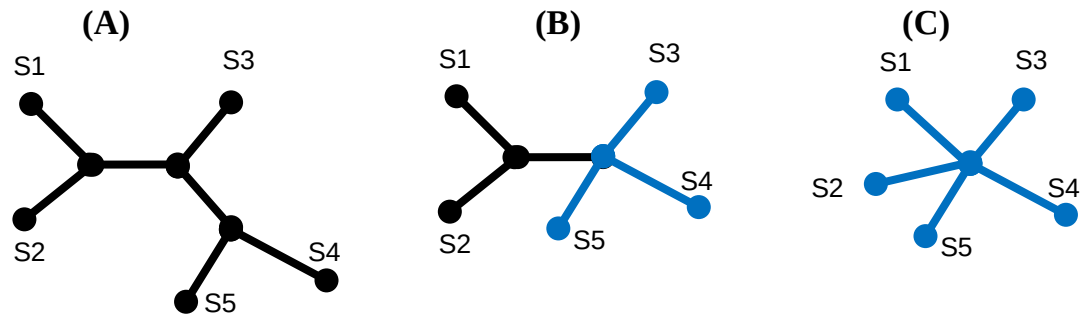
làm phép cộng (với 2 lần nạp dữ liệu vào thanh ghi và 1 lần đọc từ thanh ghi ra c). Tức là, với ví dụ này, AVX2 đem lại tăng tốc 8 lần.

Tuy vậy, tối ưu chương trình bằng SIMD là việc không hề đơn giản do cần lập trình tùy biến theo tập lệnh mở rộng và sử dụng ngôn ngữ lập trình bậc thấp [44,66]. Các trình biên dịch hiện đại tuy có tính năng tự động chuyển đổi tính toán về dạng vector nhưng không đủ linh hoạt cho các tình huống cụ thể. Với ngôn ngữ lập trình C++, là ngôn ngữ chúng tôi cài đặt UFBoot2, có nhiều thư viện đã được xây dựng để cho phép người lập trình sử dụng SIMD một cách tường minh và dễ dàng [44,66,89]. Chúng tôi sử dụng thư viện VCL của tác giả Agner Fog (thông tin chi tiết cung cấp ở địa chỉ website: https://www.agner.org/optimize/vectorclass.pdf) bởi thư viện này dễ tích hợp vào mã nguồn UFBoot2 và nó kèm theo hướng dẫn sử dụng chi tiết và đầy đủ. UFBoot2 lưu log-likelihood của mỗi vị trí sắp hàng trong một biến thực dấu phẩy động 64 bit. Do đó, sử dụng tập lệnh SSE cho phép tính toán vector chứa log-likelihood của 2 vị trí sắp hàng, tập lệnh AVX cho phép tính toán vector chứa log-likelihood của 4 vị trí sắp hàng, dẫn đến tăng tốc lý thuyết 2 lần hoặc 4 lần so với cài đặt không sử dụng SIMD.

2.4.2 Cải tiến để xử lý đỉnh đa phân tốt hơn

Các phương pháp xây dựng cây tiến hóa luôn giả sử cây có dạng nhị phân (nghĩa là mỗi đỉnh trong luôn kết nối với ba cạnh khác). Đôi khi, dữ liệu đầu vào không cho ta đủ thông tin tiến hóa, dẫn đến việc biểu diễn tốt nhất lại là một cây có chứa đỉnh đa phân (cây tồn tại ít nhất một cạnh trong có độ dài bằng 0) (Hình 2.4B,C). Khi rơi vào trường hợp xấu nhất, tất cả các cạnh trong đều có độ dài bằng 0, cây được gọi là cây hình sao (Hình 2.4C). Các phương pháp xây dựng cây tiến hóa khác nhau cho kết quả khác nhau khi tìm dạng nhị phân cho đỉnh đa phân. Khi phân giải đỉnh đa phân có thể có nhiều cấu trúc nhị phân tối ưu ngang nhau [94]. Vì UFBoot (và các tiếp cận bootstrap khác) chỉ lưu một cây duy nhất tốt nhất cho mỗi sắp hàng bootstrap, nó có

thể gán giá trị hỗ trợ bootstrap quá cao cho các cạnh ngắn [76], tức đặt độ tin cậy cao vào cạnh không tồn tại.



Hình 2.4. Minh họa cấu trúc cây (A) không có đỉnh đa phân nào, (B) có 1 đỉnh đa phân và (C) hình sao.

Để khắc phục thiếu sót này UFBoot2 đã thực hiện kỹ thuật sau đây. Thay vì chọn cây bootstrap với điểm số RELL cao nhất cho mỗi sắp hàng bootstrap, UFBoot2 sẽ chọn ngẫu nhiên một trong số các cây có điểm số RELL khác biệt không quá ε_{boot} (mặc định là 0.5) so với RELL cao nhất. Nhờ đó, UFBoot2 sẽ không gán giá trị hỗ trợ bootstrap quá cao cho các cạnh phân giải đỉnh đa phân.

2.4.3 Cải tiến để giảm ảnh hưởng của vi phạm mô hình

Kết quả phân tích tiến hóa theo tiêu chuẩn ML có thể bị ảnh hưởng bởi vấn đề vi phạm giả thiết mô hình tiến hóa.

Ta giả sử dữ liệu sắp hàng đã được tiến hóa từ một cây đúng $T^{đúng}$ theo một mô hình tiến hóa đúng $M^{đúng}$. Việc xây dựng cây tiến hóa là tìm một cây $T^{tốt\ nhất}$ gần nhất có thể với cây $T^{đúng}$. Khi xây dựng cây tiến hóa (và phân tích bootstrap) theo tiêu chuẩn ML, một trong những bước đầu tiên là xác định mô hình tiến hóa M (do trong thực tế ta không biết $M^{đúng}$). M được chọn từ một tập định trước các mô hình tiến hóa. Các mô hình khác nhau có số lượng tham số tự do khác nhau. Trong quá trình xây dựng cây, ta sẽ duyệt các giá trị khác nhau cho các tham số tự do của mô hình M này để tìm bộ giá trị tham số cho độ hợp lý cao nhất.

Khi mô hình M được chọn là khác $M^{đúng}$, ta gọi nó là mô hình vi phạm giả thiết.

Luận án xem xét hai loại vi phạm giả thiết mô hình tiến hóa: (i) Vi phạm nhẹ (ít): là mô hình M để xây dựng cây có ít tham số tự do hơn $M^{đúng}$ nhưng vẫn đảm bảo tính đồng nhất (hoặc không đồng nhất) về tốc độ biến đổi giữa các vị trí như giả thiết của $M^{đúng}$ (ii) Vi phạm nghiêm trọng (nhiều): là mô hình M để xây dựng cây không đảm bảo được tính đồng nhất (hoặc không đồng nhất) về tốc độ biến đổi giữa các vị trí như giả thiết của $M^{đúng}$. Ví dụ: $M^{đúng} = GTR + \Gamma$ thì $M = JC + \Gamma$ là vi phạm nhẹ, còn $M = JC$ là vi phạm nghiêm trọng.

Minh và cộng sự [56] đã chỉ ra rằng vi phạm mô hình nhiều sẽ làm tăng giá trị hỗ trợ bootstrap của UFBoot. Để giải quyết vấn đề này UFBoot2 cung cấp một tùy chọn để tiến hành một bước tinh chỉnh tối ưu khi tìm kiếm cây trên sắp hàng gốc hoàn tất. Theo đó, các cây RELL tốt nhất tiếp tục được tối ưu nhờ tìm kiếm leo đồi dùng NNI dựa trực tiếp trên sắp hàng bootstrap tương ứng. Do đó, bước bổ sung này hoạt động như SBS, nhưng là dạng tìm kiếm cây nhanh nhằm tiết kiệm thời gian. Các giá trị hỗ trợ bootstrap sau đó được tóm tắt từ các cây bootstrap đã tinh chỉnh tối ưu. Sau đây, chúng tôi gọi tùy chọn này là UFBoot2+NNI, có thể được bật lên khi dùng IQ-TREE thông qua tùy chọn "-bnni".

Việc khảo sát ảnh hưởng của mô hình tiến hóa vi phạm giả thiết tới độ chuẩn xác bootstrap có thể được thực hiện thông qua thực nghiệm mô phỏng. Thực nghiệm có quy trình như đã mô tả trong phần 1.5.3.2, ta lựa chọn $M^{đúng}$ và M để phản ánh vi phạm giả thiết mô hình.

2.4.4 Cải tiến mở rộng để phân tích sắp hàng các bộ gen

Các nghiên cứu gần đây về phân tích quan hệ tiến hóa thường dựa trên nhiều gen để suy ra cây tiến hóa, hướng này được gọi là phylogenomics. Để hỗ trợ phylogenomics, UFBoot2 đề xuất và cài đặt một số chiến lược sinh sắp hàng bootstrap như sau: (i) lấy mẫu có hoàn lại các vị trí sắp hàng trong từng gen của sắp hàng gốc (gọi là lấy mẫu theo vị trí sắp hàng; được đặt làm tùy chọn mặc định trong UFBoot2),

(ii) lấy mẫu có hoàn lại các gen thay vì các vị trí trên sắp hàng (gọi là lấy mẫu theo gen; được gọi thông qua tham số dòng lệnh "-bsam GENE") và (iii) lấy mẫu có hoàn lại các gen và sau đó tiếp tục lấy mẫu có hoàn lại các vị trí sắp hàng trong mỗi gen (gọi là lấy mẫu theo gen-vị trí; được gọi thông qua tham số dòng lệnh "-bsam GENESITE") [25]. Chiến lược (i) đảm bảo số lượng vị trí sắp hàng cho tất cả các gen trên các sắp hàng bootstrap giữ nguyên như trong sắp hàng gốc, trong khi đó các chiến lược (ii) và (iii) sẽ dẫn đến số lượng vị trí trên các sắp hàng bootstrap khác đi.

2.5 Thực nghiệm và kết quả

Luận án khảo sát hiệu năng của các phương pháp: UFBoot gốc (phiên bản 0.9.6, phát hành vào ngày 20 tháng 10 năm 2013), UFBoot2, UFBoot2+NNI, SBS (bootstrap chuẩn 1000 bản sao bootstrap bằng IQ-TREE) và RBS (sử dụng tiêu chuẩn bootstopping của RAxML) trên các bộ dữ liệu mô phỏng và các bộ dữ liệu sinh học để kiểm tra hiệu quả của các đề xuất cải tiến. Thứ tự các kết quả trong phần này được trình bày giống với thứ tự các đề xuất cải tiến trong phần 2.4.

2.5.1 Thời gian tính toán

Luận án thực hiện đánh giá thời gian chạy trên bộ dữ liệu thực (với các thông số tóm tắt trong Bảng 2.1) được thu thập từ cơ sở dữ liệu trực tuyến TreeBASE [73] bởi Nguyen và cộng sự [59].

Bảng 2.1. Thông tin bộ dữ liệu thực từ TreeBASE.

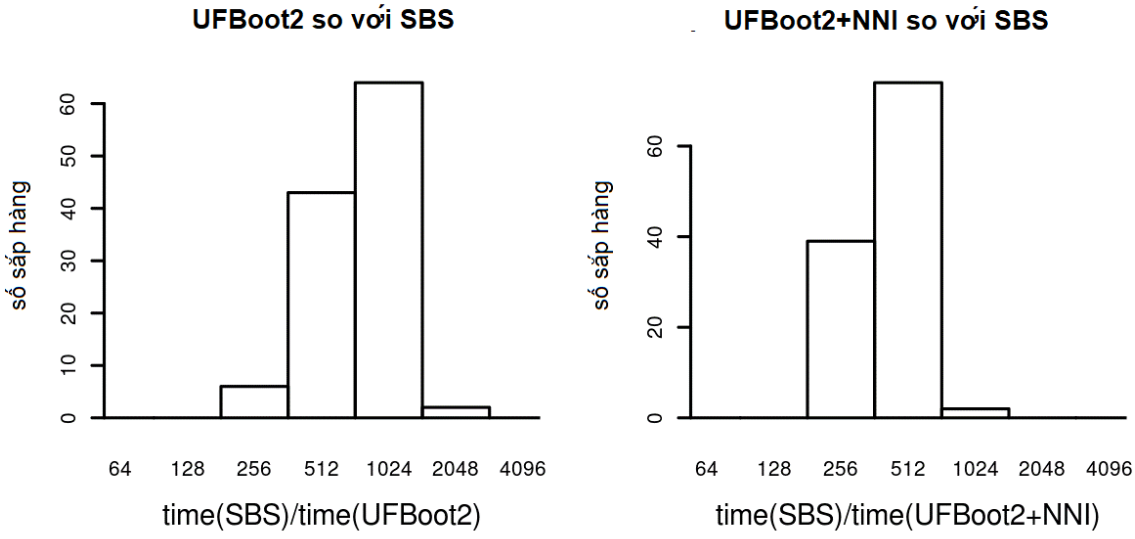
Thông số	**Loại dữ liệu**	
	DNA	**protein**
Số sắp hàng	70	45
Số taxa	201-767 (trung vị: 233)	50-194 (trung vị: 78)
Số vị trí sắp hàng	976-61199 (trung vị: 3328.5)	126-22426 (trung vị: 696)

Theo đó, tác giả tải về từ TreeBASE các sắp hàng đa chuỗi thỏa mãn ba tiêu chí sau: (1) sắp hàng DNA phải có từ 200 đến 800 chuỗi, sắp hàng protein phải có từ 50 đến 600 chuỗi; (2) số vị trí của sắp hàng DNA không nhỏ hơn 4 lần số chuỗi của nó, số vị trí của sắp hàng protein không nhỏ hơn 2 lần số chuỗi của nó; (3) tỉ lệ vị trí có

trạng thái không xác định/vị trí trống không vượt quá 70% tổng số vị trí của sắp hàng. Sau đó, tác giả lọc bỏ các chuỗi trùng lặp để đảm bảo mỗi sắp hàng chỉ chứa các chuỗi phân biệt. Bộ dữ liệu thu được có 70 sắp hàng DNA và 45 sắp hàng protein.

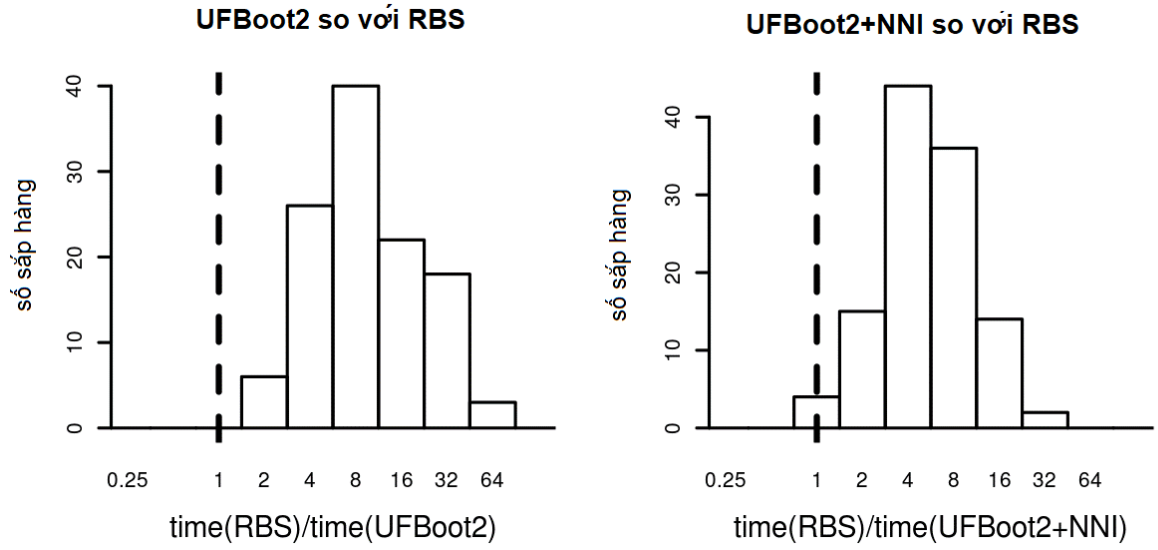
Các dòng lệnh được sử dụng để thực hiện các phương pháp bootstrap được cung cấp trong Bảng P1 (phần Phụ lục).

Hình 2.5, Hình 2.6, Hình 2.7 phác họa phân bố của tỉ lệ thời gian chạy giữa SBS / RBS / UFBoot và UFBoot2 / UFBoot2+NNI. Hàm time() kí hiệu cho thời gian thực thi của từng phương pháp.

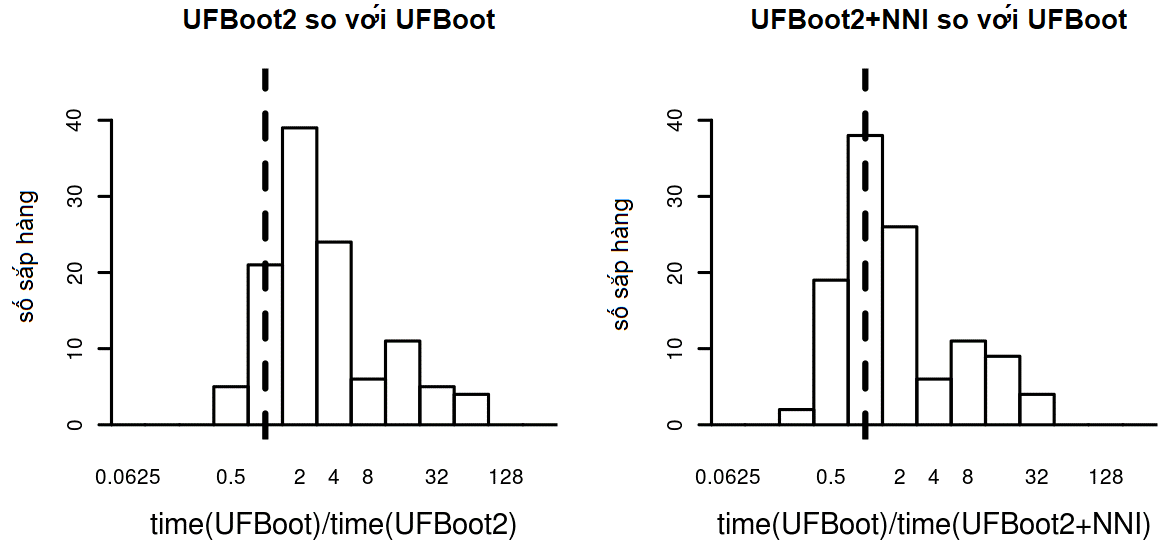


Hình 2.5. Phân bố của tỉ lệ thời gian chạy giữa SBS và UFBoot2 (trái) và giữa SBS và UFBoot2 + NNI (phải) trên 115 sắp hàng TreeBASE.

UFBoot2 nhanh hơn trung bình 2.4 lần (tối đa: 77.3) so với phiên bản UFBoot 0.9.6 (xem Hình 2.7 trái). UFBoot2 và UFBoot2+NNI cho tăng tốc 778 lần (trung vị; miền tỉ lệ: 200-1848) và 424 lần (miền tỉ lệ: 233-749) so với SBS. So với RBS, UFBoot2 và UFBoot2+NNI cho tăng tốc 8.4 lần (trung vị; miền tỉ lệ: 1.5-51.2) và 5.0 lần (miền tỉ lệ: 0.8-32.6). Vì vậy, UFBoot2+NNI chậm hơn hai lần (trung vị) so với UFBoot2.



Hình 2.6. Phân bố của tỉ lệ thời gian chạy giữa RBS và UFBoot2 (trái) và giữa RBS và UFBoot2 + NNI (phải) trên 115 sắp hàng TreeBASE.



Hình 2.7. Phân bố của tỉ lệ thời gian chạy giữa UFBoot gốc và UFBoot2 (trái) và giữa UFBoot gốc và UFBoot2 + NNI (phải) trên 115 sắp hàng TreeBASE.

2.5.2 Tỉ lệ dương tính giả

Để đánh giá hiệu quả xử lý đỉnh đa phân của ý tưởng đề xuất trong 2.4.2, luận án khảo sát tỉ lệ dương tính giả (false positive) khi phân tích dữ liệu mô phỏng tiến hóa từ cây hình sao. Quy trình mô phỏng giống như trình bày trong phần 1.5.3.2,

trong đó, ta chọn cây đúng là cây hình sao. Do tất cả các cạnh trong của cây xây dựng được đều không tồn tại trên cây đúng, một phương pháp bootstrap tốt sẽ không tính ra giá trị hỗ trợ bootstrap cao (≥95%) trên dữ liệu này, nghĩa là không tạo kết quả dương tính giả.

Cụ thể, chúng tôi tạo dữ liệu mô phỏng với cùng thiết kế như đề xuất trong [76]. Chúng tôi sử dụng Seq-Gen 1.3.2x [69] để sinh 100 sắp hàng DNA, mỗi sắp hàng chứa 15000 vị trí sắp hàng, tiến hóa từ cây đúng hình sao có 4 lá và 4 cạnh nối với lá có chiều dài 0.05, theo mô hình JC. Đối với mỗi sắp hàng gốc, chúng tôi đã cho UFBoot2 phân tích bằng mô hình JC và GTR+Γ, mỗi phân tích làm bootstrap 1000 bản sao và tới 1000 vòng lặp tìm kiếm (cài đặt trong IQ-TREE thông qua tùy chọn "-bcor 1"). Với mỗi phân tích này, chúng tôi khảo sát 2 chế độ: (i) bật tính năng cải tiến để xử lý đỉnh đa phân với $\varepsilon_{boot} = 50$ (do sắp hàng có nhiều vị trí) và (ii) tắt cải tiến này, nghĩa là chọn cây có điểm số RELL cao nhất làm cây bootstrap. Mỗi chế độ được thực thi 2 lượt sử dụng 2 hạt giống ngẫu nhiên khác nhau (thông qua tùy chọn "- seed 123456" và "- seed 654321").

Simmons và Norton [76] đã chỉ ra rằng SBS và RBS đôi khi dẫn đến dương tính giả , trong khi UFBoot không bao giờ hỗ trợ cạnh như vậy (giá trị hỗ trợ bootstrap ≤88%). Cần lưu ý rằng, phiên bản UFBoot khảo sát trong [76] là bản đã có cải tiến để xử lý đỉnh đa phân và là bản mới hơn UFBoot nguyên thủy trong [56].

Sử dụng UFBoot2 phân tích dữ liệu mô phỏng từ cây đúng hình sao, chúng tôi thu được kết quả tóm tắt trong Bảng 2.2. Trong đó, cột max lưu trung bình cộng 2 giá trị hỗ trợ bootstrap cực đại trên từng hạt giống ngẫu nhiên. Cột min lưu trung bình cộng 2 giá trị cực tiểu tính tương tự. Kết quả đã khẳng định hiệu quả của ý tưởng chọn cây bootstrap là cây ngẫu nhiên trong số cây có điểm số RELL khác biệt không quá ε_{boot} so với RELL cao nhất so với việc chọn cây có RELL cao nhất. Khi bật cải tiến này, UFBoot2 gán giá trị hỗ trợ bootstrap rất thấp cho cạnh không tồn tại. Miền giá trị thu được (30%-38.5%) sát với xác suất chọn ngẫu nhiên 1 trong 3 cấu trúc nhị

phân để phân giải cây hình sao 4 lá (xác suất bằng 1/3). Khi tắt cải tiến này, nó có thể cho giá trị hỗ trợ bootstrap lớn hơn xác suất này, tuy nhiên vẫn nhỏ hơn ngưỡng 95%.

Bảng 2.2. Tóm tắt giá trị hỗ trợ bootstrap cho cạnh đúng không tồn tại của UFBoot2 khi bật và tắt cải tiến xử lý đỉnh đa phân trên dữ liệu mô phỏng từ cây đúng hình sao.

Loại phân tích	**Giá trị hỗ trợ bootstrap cho cạnh trong**	
	min	**max**
UFBoot2 bật cải tiến với JC	30	38.5
UFBoot2 bật cải tiến với GTR+Γ	30	38.5
UFBoot2 tắt cải tiến với JC	35	89.5
UFBoot2 tắt cải tiến với GTR+Γ	29	87

2.5.3 Độ chuẩn xác của ước lượng bootstrap

Luận án thực hiện lại thực nghiệm với dữ liệu mô phỏng PANDIT [56] để so sánh độ chuẩn xác bootstrap của UFBoot2 và UFBoot2+NNI với SBS và RBS. Dữ liệu mô phỏng bao gồm 5690 sắp hàng DNA (DOI 10.5281/zenodo.854445) tạo ra bằng công cụ Seq-Gen [69], trong đó các tham số mô hình và cây đúng được suy luận từ các sắp hàng gốc tải về từ cơ sở dữ liệu PANDIT [93]. Bảng 2.3 tóm tắt các thông số của bộ dữ liệu DNA mô phỏng PANDIT. Chi tiết các bước sinh dữ liệu mô phỏng trình bày trong **Thuật toán 2.4**. Trong đó, bước 4 là để sau đó làm phân tích bootstrap trên dữ liệu này có thể thử vi phạm mô hình nhẹ và vi phạm mô hình nghiêm trọng.

Bảng 2.3. Thông tin bộ dữ liệu DNA mô phỏng PANDIT.

Thông số	**Giá trị**
Số sắp hàng	5690
Số taxa	4-403 (trung vị: 11)
Số vị trí sắp hàng	33-6891 (trung vị: 600)

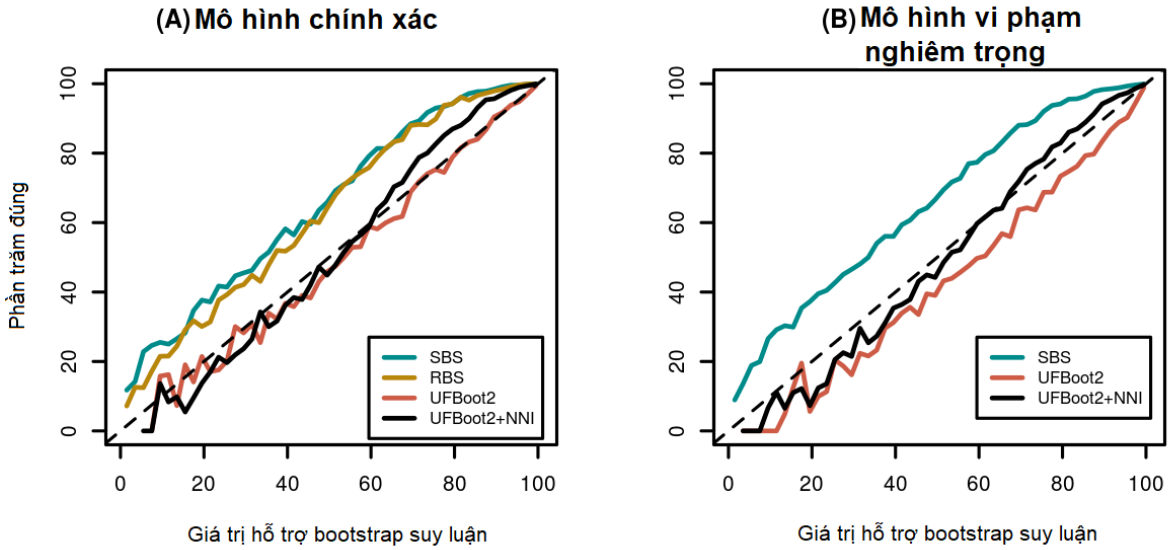
Sử dụng khái niệm độ chuẩn xác đã được giải thích trong phần 1.5.3.2, chúng tôi tóm tắt kết quả trong Hình 2.8 (trục y biểu diễn $f_Z(x)$). Trong mỗi điểm (x, y) trên đồ thị biểu diễn độ chuẩn xác của phương pháp bootstrap Z, x là một giá trị hỗ trợ bootstrap do Z gán, còn y đo xác suất các cạnh được Z gán giá trị hỗ trợ bootstrap x sẽ là cạnh đúng.

Thuật toán 2.4. Phương pháp sinh bộ dữ liệu DNA mô phỏng PANDIT

Bắt đầu

1) Tải các sắp hàng DNA chứa ít nhất 4 taxa từ trang web cơ sở dữ liệu PANDIT: thu được 6491 sắp hàng thực
2) Xóa bớt các sắp hàng ngắn (số cột nhỏ hơn 3 lần số hàng): thu được 6222 sắp hàng thực
3) Với mỗi sắp hàng thực A:
   - chọn mô hình tiến hóa phù hợp nhất M nhờ công cụ ModelTest [67]
   - xây dựng cây T là cây tốt nhất theo tiêu chuẩn ML nhờ IQ-TREE bằng mô hình M đã chọn
   - coi T là cây đúng, dùng Seq-gen để sinh sắp hàng mô phỏng A' có cùng kích thước như A và theo các tham số mô hình trong M
   - chèn các vị trí sắp hàng (cột) trống trong A vào A'
4) Chọn từ 6222 sắp hàng mô phỏng các sắp hàng có mô hình M phức tạp hơn $JC + \Gamma$: thu được 5690 sắp hàng mô phỏng

Kết thúc



Hình 2.8. Độ chuẩn xác của bootstrap chuẩn (SBS), bootstrap nhanh của RAxML (RBS), UFBoot2 và UFBoot2 với bước tinh chỉnh tối ưu (UFBoot2+NNI) cho (A) mô hình chính xác và (B) vi phạm mô hình nhiều. Trục y biểu diễn phần trăm các cạnh có giá trị hỗ trợ bootstrap x (trong tất cả các cây xây dựng được) có mặt trong cây đúng.

Nếu xây dựng cây bằng mô hình tiến hóa không vi phạm giả thiết thì SBS, RBS và UFBoot2+NNI gán giá trị hỗ trợ bootstrap thấp cho các cạnh, phương pháp sau ít

sai lệch hơn phương pháp trước (Hình 2.8A, đồ thị phía trên đường chéo). Xu hướng gán giá trị hỗ trợ bảo thủ của SBS và RBS đã khẳng định các nghiên cứu trước đây [39,56]. Trong khi đó, UFBoot2 cho các giá trị hỗ trợ bootstrap gần như không chệch (Hình 2.8A, đồ thị gần với đường chéo), nghĩa là gần như khớp với xác suất là cạnh đúng. Do đó, UFBoot2 có độ chuẩn xác giống bản UFBoot gốc [56].

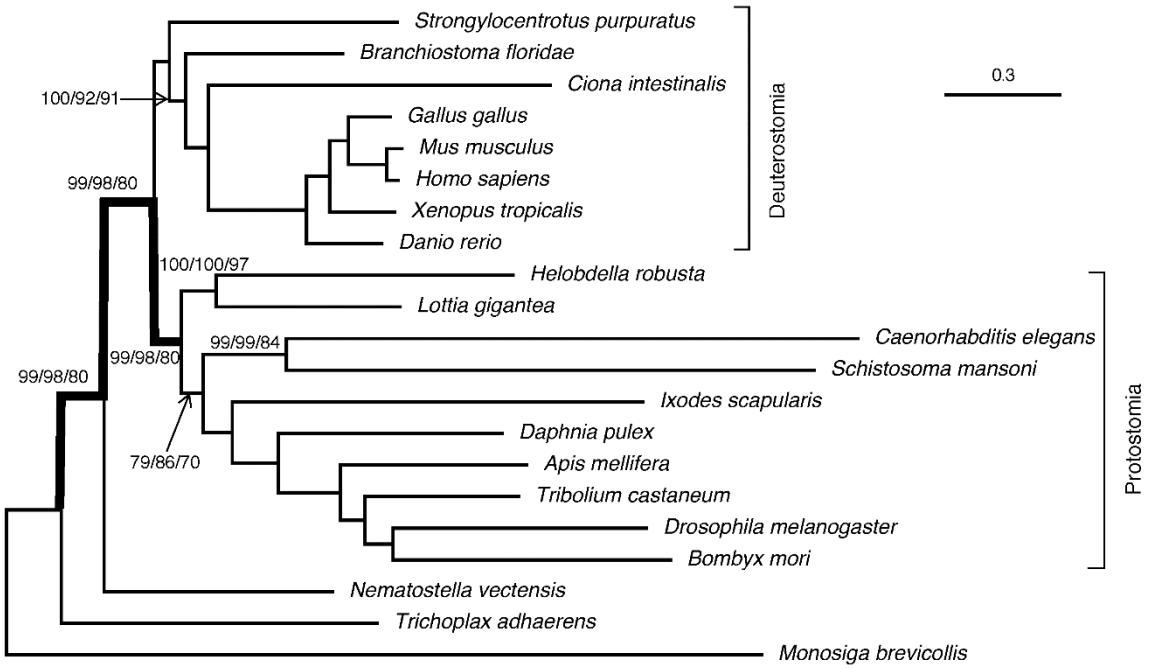
Vi phạm mô hình nhiều không ảnh hưởng đến SBS (Hình 2.8B; không có đồ thị của RBS vì RAxML không hỗ trợ mô hình đơn giản để kiểm tra vi phạm mô hình). Tuy nhiên, UFBoot2 (cũng giống như UFBoot) đã gán giá trị hỗ trợ bootstrap cao cho các cạnh (Hình 2.8B, đồ thị nằm dưới đường chéo), trong khi đó UFBoot2+NNI có giá trị hỗ trợ bootstrap chỉ thấp hơn xác suất đúng một chút (Hình 2.8B, đồ thị gần đường chéo). Do đó, UFBoot2+NNI giúp giải quyết vấn đề giá trị hỗ trợ bootstrap cao của UFBoot2 khi xây dựng cây bằng mô hình vi phạm nhiều.

2.5.4 Khả năng phân tích sắp hàng bộ gen

Để khảo sát ảnh hưởng của ba chiến lược lấy mẫu cho phân tích sắp hàng bộ gen (phần 2.4.4), luận án phân tích lại dữ liệu metazoan trong đó có 21 loài, 225 gen và tổng số 171077 vị trí axít amin [72]. Hình 2.9 trình bày cây ML xây dựng bằng IQ-TREE theo mô hình phân hoạch edge-unlinked [9]. Mô hình này cho phép mỗi phân hoạch (gen) sử dụng một tập độ dài cạnh khác nhau. Cây này đã khẳng định lại được các kết quả trước đó [72] và khôi phục được nhóm loài Protostomia [86]. Tuy nhiên, cây cũng cho thấy sự khác biệt giữa các chiến lược lấy mẫu khác nhau: trong khi lấy mẫu theo vị trí và lấy mẫu theo gen thu được các giá trị hỗ trợ bootstrap cao (>95%) cho các cạnh nằm trên xương sống của cây (Hình 2.9; đường nét đậm), thì lấy mẫu theo gen-vị trí lại cho các giá trị hỗ trợ bootstrap thấp (80%).

Mở rộng khảo sát với 14 bộ dữ liệu sinh học khác [7,12,17,40,53,62,68,72,77,80], luận án quan sát được nhiều khác biệt hơn giữa các chiến lược lấy mẫu (dữ liệu không trình bày ở đây). Đặc biệt, trên một số bộ dữ liệu, có những cạnh gần như không nhận được hỗ trợ nào (≤10%) từ một chiến lược lấy

mẫu nhưng lại có hỗ trợ cao (≥95%) từ hai chiến lược lấy mẫu kia. Tuy nhiên, không có dấu hiệu nào về việc một chiến lược luôn luôn cho giá trị hỗ trợ bootstrap thấp.



Hình 2.9. Cây hợp lý nhất (ML) xây dựng theo mô hình phân hoạch edge-unlinked. Các con số gắn với các cạnh là các giá trị hỗ trợ bootstrap do UFBoot2 gán cho cạnh với các chiến lược lấy mẫu: theo vị trí, theo gen, và gen-vị trí (số bị ẩn đi nếu cả 3 phương pháp đều cho giá trị hỗ trợ bootstrap 100%).

Từ những phát hiện nói trên, luận án khuyến cáo người dùng áp dụng tất cả các chiến lược lấy mẫu. Nếu các giá trị hỗ trợ bootstrap thu được tương tự nhau thì kết quả đáng tin cậy hơn.

2.6 Kết luận chương

Tính toán likelihood là nút thắt chính trong thời gian chạy của tất cả các phần mềm ML vì nó nằm ở phần lõi của tất cả các phân tích. Thuật toán pruning [19,20] giúp tính toán hiệu quả likelihood của cây tiến hóa, nhưng nó chưa đủ nhanh cho dữ liệu kích thước lớn. Do đó, luận án đề xuất phiên bản sửa đổi thuật toán của Felsenstein, đặt tên là *thuật toán pruning nhanh*. Thuật toán sửa đổi đã được cài đặt trước đây trong phần mềm RAxML nhưng chưa bao giờ được công bố chính thức. Sửa đổi này khai thác tính chất thuận nghịch thời gian của các mô hình tiến hóa

thường được sử dụng, dẫn đến tốc độ lý thuyết tăng lên 4 lần (đối với dữ liệu DNA) hoặc 20 lần (đối với dữ liệu protein) khi ước lượng độ dài cạnh. Lưu ý rằng thuật toán đề xuất tăng tốc chỉ thông qua việc thay đổi đại lượng lưu trữ trung gian dựa trên khai thác các tính chất của các ma trận liên quan nên nó luôn đảm bảo tính đúng đắn của giá trị likelihood tính được.

Để thấy hiệu quả của *thuật toán pruning nhanh,* chúng tôi vận dụng nó trong tính toán likelihood cây của bài toán phân tích bootstrap tiến hóa ML và đề xuất phương pháp UFBoot2 dựa trên UFBoot. UFBoot2 cải thiện đáng kể tốc độ so với UFBoot (nhanh hơn trung bình 2.4 lần).

Sau cải tiến về tốc độ, chương này trình bày thêm 3 cải tiến mà luận án đề xuất trong phương pháp UFBoot2. UFBoot2 cải thiện được độ chuẩn xác của giá trị hỗ trợ bootstrap so với UFBoot khi có vi phạm giả thiết mô hình nghiêm trọng. Hơn nữa, UFBoot2 có các cải tiến để xử lý đỉnh đa phân tốt hơn và có mở rộng để phân tích sắp hàng các bộ gen. Nói chung, vì SBS, RBS và UFBoot2+NNI có điểm yếu chung là gán giá trị hỗ trợ thấp, cần phải có các nghiên cứu sâu hơn để hiểu được xu hướng chệch/không chệch khác nhau của các phương pháp bootstrap trong phân tích tiến hóa hiện nay.

Luận án kết luận rằng UFBoot2 và UFBoot2+NNI là những phương pháp nhanh thay thế cho các tiếp cận bootstrap khác. Khi mô hình không sai hoặc sai ít, ta có thể dùng các giá trị hỗ trợ bootstrap UFBoot2 với ý nghĩa không chệch đã đề xuất cho UFBoot [56]. Nghĩa là, người dùng có thể tin tưởng các cạnh có giá trị hỗ trợ bootstrap từ UFBoot2 $\geq 95\%$. Người dùng nên sử dụng các phương pháp phát hiện vi phạm mô hình [28,60,91] trước khi làm phân tích bootstrap. Khi có khả năng cao xảy ra vi phạm mô hình thì người dùng cần sử dụng UFBoot2+NNI.

Các kết quả nghiên cứu của chương này đã được công bố trong bài báo đăng trên tạp chí Molecular Biology and Evolution năm 2018 (công trình khoa học số 2).

Chương 3 PHƯƠNG PHÁP MỚI MPBOOT GIẢI NHANH BÀI TOÁN XÂY DỰNG CÂY BOOTSTRAP TIẾN HÓA THEO TIÊU CHUẨN TIẾT KIỆM NHẤT

Chương này tập trung vào bài toán xây dựng cây bootstrap tiến hóa theo tiêu chuẩn tiết kiệm nhất và đề xuất phương pháp MPBoot.

3.1 Giới thiệu

Cực tiểu số lượng biến đổi hay còn gọi là tiết kiệm nhất (maximum parsimony – MP) là tiêu chuẩn được sử dụng rộng rãi trong xây dựng cây tiến hóa [22 và các tài liệu nó tham khảo]. Vì tính điểm MP cho một cây ít phức tạp hơn và chi phí tính toán thấp hơn so với tính likelihood, các phương pháp để xây dựng cây MP đã được áp dụng cho các bộ dữ liệu lớn [29,31]. Tuy nhiên, tính toán giá trị hỗ trợ bootstrap cho cạnh của các cây MP vẫn tiêu tốn nhiều thời gian, đặc biệt là đối với các bộ dữ liệu lớn. Ngoài những hạn chế về thời gian chạy, bootstrap chuẩn cũng được chứng minh là "bảo thủ" [39] (xem giải thích chi tiết trong phần 1.5.3.2): giá trị hỗ trợ bootstrap tính theo bootstrap chuẩn thường thấp hơn xác suất để cạnh là cạnh đúng.

Chương này giới thiệu MPBoot, một phương pháp mới để tìm nhanh lời giải chấp nhận được cho bootstrap theo tiêu chuẩn MP. MPBoot được phát triển từ ý tưởng của UFBoot – là phương pháp bootstrap nhanh cho tiêu chuẩn ML [56]. Để phù hợp với MP, MPBoot vận dụng thêm các kỹ thuật cho tìm kiếm cây như cắt và ghép cây con (subtree pruning and regrafting – SPR) và ratchet [61]. Dưới đây luận án sẽ trình bày phương pháp MPBoot và thực nghiệm so sánh MPBoot với các chương trình tiêu biểu cho phân tích tiến hóa sử dụng tiêu chuẩn MP là TNT [31] và PAUP* [84].

3.2 Xây dựng cây tiến hóa theo tiêu chuẩn MP

Ký hiệu A^{data} là sắp hàng của n chuỗi và m vị trí sắp hàng (sau khi loại bỏ các vị trí hằng). m vị trí này được nhóm thành các mẫu-vị trí $D_1, D_2, \dots, D_k$ với tần suất

tương ứng là $d_1, d_2, \ldots, d_k$. Điểm MP cho cấu trúc cây T khi biết A^{data} được tính bởi công thức:

$$MP(T|A^{data}) = \sum_{i=1}^{k} MP(T|D_i) \times d_i \quad (3.1)$$

trong đó $MP(T|D_i)$ là điểm MP cho cây T tại mẫu-vị trí D_i.

Với một cây T cho trước, $MP(T|D_i)$ có thể tính được một cách hiệu quả nhờ thuật toán Fitch [23] trong trường hợp chi phí biến đổi giữa các trạng thái là bằng nhau, tức sử dụng *ma trận chi phí đều* (uniform cost matrix, xem Bảng 1.3A). Ma trận chi phí đều có ưu điểm là đơn giản về mặt khái niệm nhưng lại không thực sự có ý nghĩa sinh học. Ví dụ, một quan sát đã trở nên phổ biến trong sinh học phân tử là với các chuỗi nucleotide thì các biến đổi cùng nhóm (transitions) xảy ra thường xuyên hơn các biến đổi chéo nhóm (transversions). Do đó, sẽ hợp lý hơn nếu ta đặt chi phí biến đổi cùng nhóm thấp hơn. Trường hợp này, ta có *ma trận chí phí không đều* (non-uniform cost matrix, xem Bảng 1.3B). Khi dùng một ma trận chi phí không đều, ta không thể tiếp tục sử dụng cách giải nghĩa điểm MP như là số lượng biến đổi tối thiểu nữa. Với ma trận chi phí không đều, ta dùng thuật toán Sankoff [74] để tính $MP(T|D_i)$. Một phương pháp tìm kiếm cây (còn gọi là xây dựng cây) theo tiêu chuẩn MP có mục tiêu tìm một cây có điểm MP nhỏ nhất. Bài toán tìm cây MP tốt nhất thuộc lớp NP-đầy đủ [32], do đó, cần sử dụng các heuristic cho tìm kiếm cây.

3.3 Đề xuất thuật toán MPBoot

Để giảm thiểu thời gian tính toán, các tính năng chính của phương pháp MPBoot là (i) lấy mẫu cây từ không gian cây chỉ dựa trên sắp hàng gốc thay vì tiến hành các lượt tìm kiếm cây độc lập riêng biệt cho từng sắp hàng bootstrap và (ii) nhanh chóng tính điểm MP của các cây mẫu trên tất cả các sắp hàng bootstrap. Sau đây, luận án mô tả các thành phần chính và luồng xử lý tổng thể của MPBoot.

3.3.1 Lấy mẫu cây trên sắp hàng gốc

Trong khi bootstrap chuẩn thực hiện một thủ tục tìm kiếm cây độc lập cho mỗi sắp hàng bootstrap, MPBoot không làm như vậy. Trước tiên, MPBoot tạo ra một tập các sắp hàng bootstrap và sau đó tìm kiếm trên không gian cây dựa trên sắp hàng gốc. Các cây được duyệt (còn gọi là lấy mẫu) trong quá trình tìm kiếm cây này được coi là cây MP tiềm năng cho mỗi sắp hàng bootstrap.

Thuật toán MPBoot tìm kiếm cây MP cho sắp hàng gốc hoạt động nhờ chỉnh sửa dần dần một tập ứng viên *C* chứa các cây tối ưu cục bộ khác nhau. Để khởi tạo tập ứng viên *C*, MPBoot tạo ra 100 cây tối ưu cục bộ theo tiêu chuẩn MP nhờ chiến lược xây dựng cây từng bước ngẫu nhiên (randomized stepwise addition) [88] theo sau là tìm kiếm leo đồi với kỹ thuật SPR [81]. Thuật toán sắp xếp các cây này tăng dần theo điểm MP của chúng, rồi chọn 5 cây phân biệt đầu tiên để tạo tập ứng viên ban đầu. Ý tưởng cho tập ứng viên *C* được lấy cảm hứng từ thuật toán di truyền [41]. Nó sẽ duy trì một quần thể cây để đảm bảo tính đa dạng. Trong quá trình tìm kiếm cây, *C* sẽ được cập nhật liên tục bằng các cây tốt hơn. Điều này hoàn tất *bước khởi đầu* của MPBoot.

Trong *bước khám phá* tiếp sau đó, MPBoot sẽ luân phiên giữa phép xáo trộn cấu trúc cây (perturbation) và phép leo đồi (hill-climbing). Việc này sẽ được lặp lại nhiều lần nhằm thoát khỏi cực trị địa phương trong không gian tìm kiếm cây.

Phép xáo trộn cấu trúc cây: MPBoot trước tiên chọn ngẫu nhiên một cây T_C từ tập ứng viên *C*. Sau đó, T_C bị xáo trộn theo một trong 2 cách sau: (i) thực hiện trao đổi ngẫu nhiên hàng xóm gần nhất (NNI ngẫu nhiên) trên 50% cạnh trong chọn ngẫu nhiên để tạo ra cây T^* hoặc (ii) áp dụng parsimony ratchet [61]. Parsimony ratchet sẽ nhân đôi 50% các vị trí có thông tin parsimony trên sắp hàng gốc để sinh ra một sắp hàng xáo trộn. Sau đó, ratchet thực hiện một lượt tìm kiếm leo đồi với kỹ thuật SPR trên sắp hàng xáo trộn này, bắt đầu từ cây T_C để tìm ra một cây tối ưu cục bộ T^*. Nói

tóm lại, T^* được tạo ra bằng chiến lược xáo trộn cây (các NNI ngẫu nhiên) hoặc bằng chiến lược xáo trộn sắp hàng.

Phép leo đồi: Cây T^* sau đó đóng vai trò cây ban đầu cho tìm kiếm leo đồi SPR để tìm ra một cây tối ưu cục bộ T^{**} cho A^{data}. Nếu $MP(T^{**}|A^{data})$ nhỏ hơn hoặc bằng điểm MP lớn nhất trên A^{data} của các cây trong tập ứng viên C thì T^{**} sẽ thế chỗ cây tương ứng trong tập ứng viên. Luận án gọi phép leo đồi là thành công nếu $MP(T^{**}|A^{data})$ nhỏ hơn điểm MP nhỏ nhất trên A^{data} của các cây trong tập ứng viên. Ngược lại, luận án gọi phép leo đồi là không thành công, nghĩa là thuật toán đã không tìm thấy một cây tốt hơn.

Bước khám phá áp dụng các phép xáo trộn theo sau là phép leo đồi cho đến khi n' (làm tròn số lượng chuỗi n lên tới hàng trăm) phép tìm kiếm leo đồi cuối cùng đều không thành công. Điều này cho phép tìm kiếm kỹ hơn nếu các sắp hàng gốc có rất nhiều chuỗi. Nghĩa là, MPBoot dừng lại bởi vì nó không thể tìm được cây tốt hơn. Đến đây *bước khám phá* hoàn tất.

Các bước này sẽ được tóm tắt bằng giả mã trong **Thuật toán 3.1**.

3.3.2 Lấy mẫu điểm MP (Resampling parsimony score - REPS)

Với một cây T duyệt khi tìm kiếm leo đồi với SPR, MPBoot tính điểm của nó cho mỗi sắp hàng bootstrap $A^{bootstrap}$ và sau đó cập nhật cây bootstrap cho $A^{bootstrap}$ nếu T cho điểm MP tốt hơn. Vì việc này tiêu tốn nhiều thời gian nên ta cần tính toán hiệu quả điểm MP của T cho $A^{bootstrap}$. Để làm điều đó, luận án đã phỏng theo phương pháp RELL được đề xuất bởi Kishino và cộng sự [45] (xem thêm 2.3.2) để tính giá trị lấy mẫu điểm MP (resampling parsimony score - REPS) cho mỗi sắp hàng bootstrap. Tuy nhiên, trong khi RELL chỉ thu được log-likelihoods xấp xỉ thì REPS luôn luôn trả về điểm MP chính xác. Với cây T và các điểm số mẫu-vị trí $MP(T|D_i)$ tính trên A^{data}, điểm MP trên $A^{bootstrap}$ được tính nhanh bằng tổng có trọng số của các điểm số mẫu-vị trí:

$$MP(T|A^{bootstrap}) = \sum_{i=1}^{k} MP(T|D_i) \times d_i^{bootstrap} \quad (3.2)$$

trong đó $d_i^{bootstrap}$ là tần suất mẫu-vị trí D_i khi tạo $A^{bootstrap}$. Nhờ vậy, không cần phải lặp lại việc tính điểm MP cho mỗi mẫu-vị trí, mỗi bản sao bootstrap và mỗi cây.

3.3.3 Tăng tốc tính toán REPS

Để tăng tốc việc tính điểm MP, MPBoot sử dụng thêm hai kỹ thuật tối ưu thuật toán.

Thứ nhất, luận án đề xuất sử dụng một ngưỡng MP_{max}, sao cho điểm MP tính trên các sắp hàng bootstrap sử dụng (3.2) chỉ áp dụng cho những cây T được tìm thấy trong phép leo đồi thỏa mãn $MP(T|A^{data}) < MP_{max}$. Ban đầu, $MP_{max} = +\infty$. Sau phép leo đồi đầu tiên, thuật toán thiết lập MP_{max} là bách phân vị thứ 90 của phân bố điểm MP trên sắp hàng gốc của tất cả các cây duyệt trong phép leo đồi này. Trong các phép leo đồi tiếp theo, thuật toán chỉ xem xét các cây có điểm MP trên A^{data} nhỏ hơn MP_{max}. Điểm MP của những cây này tạo thành một phân bố sau đó sẽ được sử dụng để cập nhật MP_{max} sau mỗi phép leo đồi như trên.

Thứ hai, thuật toán dừng việc tính toán REPS cho một $A^{bootstrap}$ nếu ta không thể mong đợi rằng $MP(T|A^{bootstrap})$ sẽ nhỏ hơn điểm MP tốt nhất tính đến hiện tại $MP_{best}(A^{bootstrap})$. Để làm điều này, thuật toán sắp xếp các mẫu-vị trí D_i theo điểm MP giảm dần dựa trên cây đầu tiên xây dựng được ở *bước khởi đầu*. Điểm MP nhỏ nhất về mặt lý thuyết, $MP_{min}(D_i)$, bằng số các ký tự trạng thái riêng biệt có mặt trong D_i trừ đi 1 trong trường hợp ma trận chi phí đều. Trường hợp ma trận chi phí không đều, $MP_{min}(D_i)$ bằng độ dài của cây khung nhỏ nhất trên đồ thị chi phí. Đồ thị này có các đỉnh tương ứng các ký tự trạng thái có mặt trong D_i và trọng số cạnh bằng chi phí biến đổi giữa các trạng thái. Thuật toán dừng việc tính toán REPS nếu

$$\sum_{i=1}^{j} MP(T|D_i) \times d_i^{bootstrap} + \sum_{i=j+1}^{k} MP_{min}(D_i) \times d_i^{bootstrap} > MP_{best}(A^{bootstrap}) \quad (3.3)$$

với một số giá trị j ($1 \leq j \leq k$), bởi T không thể là cây MP cho $A^{bootstrap}$. Tổng riêng phần thứ nhất ở vế trái của (3.3) là điểm MP tính cho j mẫu-vị trí đầu tiên của $A^{bootstrap}$, còn tổng riêng phần thứ hai là cận dưới cho điểm MP của $k - j$ mẫu-vị trí còn lại. Nếu (3.3) đúng, ta biết rằng T tồi hơn cây bootstrap tốt nhất hiện tại cho $A^{bootstrap}$, mà không cần tính điểm MP của $k - j$ mẫu-vị trí còn lại.

3.3.4 Thuật toán MPBoot

Ta có thể tóm tắt toàn bộ hoạt động của MPBoot như trong **Thuật toán 3.1**. So với UFBoot (**Thuật toán 2.1**) thì MPBoot có thêm bước tinh chỉnh để tối ưu từng cây bootstrap (bước 4) dựa trên sắp hàng bootstrap tương ứng. Khác biệt này là do không gian cây theo tiêu chuẩn MP tồn tại nhiều cây có điểm số bằng nhau bởi điểm MP là số nguyên (trong khi điểm log-likelihood là số thực, không gian cây theo tiêu chuẩn ML hiếm khi có 2 cây điểm số bằng nhau). Việc lấy mẫu cây trong MPBoot chỉ dựa trên sắp hàng gốc kéo theo khả năng cao là các cây bootstrap bị trùng nhau, dẫn tới khả năng gán giá trị hỗ trợ bootstrap cao cho những cạnh không thuộc cây đúng.

MPBoot sử dụng thư viện tính toán likelihood cho phân tích tiến hóa (phylogenetic likelihood library - PLL) [24] để hỗ trợ tính toán cây MP hiệu quả. Để giảm chi phí tính toán của các phép tìm kiếm SPR, PLL sử dụng một bán kính giới hạn số đỉnh tối đa giữa cạnh ứng với việc cắt cây con và cạnh ứng với việc ghép cây con. Do đó, chúng tôi đã thử nghiệm hai phiên bản có bán kính SPR bằng 3 và 6, ký hiệu lần lượt là MPBoot SPR3 và MPBoot SPR6. Cuối cùng, mã nguồn của tất cả các tính toán cốt lõi theo tiêu chuẩn MP và tính toán REPS được tối ưu để tăng tốc nhờ khai thác công nghệ SIMD như đã trình bày trong phần 2.4.1.4.

Thuật toán 3.1. Thuật toán MPBoot

Dữ liệu vào: Sắp hàng gốc A^{data} gồm n chuỗi (taxa).

Dữ liệu ra: Tập cây bootstrap. Cây MP xây dựng cho sắp hàng gốc được gắn các giá trị hỗ trợ bootstrap cho mỗi cạnh.

Bắt đầu

1) Bước khởi đầu: Tạo B sắp hàng bootstrap, $A_1, A_2, \dots, A_B$. Với mỗi sắp hàng bootstrap A_b khởi tạo cây bootstrap $T_b := null$ và $MP(T_b|A_b) := +\infty$. Khởi tạo một tập các cây $S := \{\}$ và ngưỡng $MP_{max} := +\infty$. Khởi tạo tập ứng viên C trên A^{data} như giải thích trong phần 3.3.1.
2) Bước khám phá: Từ một cây chọn ngẫu nhiên trong tập ứng viên C, thực hiện phép xáo trộn theo sau là phép leo đồi. Mỗi khi duyệt một cây T mới thỏa mãn $MP(T|A^{data}) < MP_{max}$, thêm T vào S và tính $MP(T|A_b)$, cho các $b = 1, \dots, B$ dựa trên (3.2). Nếu $MP(T|A_b) < MP(T_b|A_b)$, cập nhật $T_b := T$. Khi phép leo đồi thực hiện xong, cập nhật MP_{max} bằng bách vị phân thứ 90 của phân bố các điểm MP của các cây trong S.
3) Điều kiện dừng: Nếu n' phép leo đồi liên tiếp đều không thành công, thực hiện bước 4. Ngược lại, quay về bước 2.
4) Bước tinh chỉnh: Với mỗi cây MP T_b $(b = 1, \dots, B)$ và sắp hàng tương ứng A_b, tiến hành một lượt tìm kiếm leo đồi dùng SPR và thay thế T_b bằng cây MP mới nếu cây mới có điểm MP tốt hơn.
5) Bước tóm tắt: Xây dựng một cây đồng thuận từ các cây bootstrap $\{T_1, T_2, \dots, T_B\}$, hoặc tính và gắn giá trị hỗ trợ bootstrap lên cây MP tốt nhất trên A^{data}.

Kết thúc

3.4 Thiết kế thực nghiệm

Luận án so sánh hiệu năng của MPBoot SPR3 và SPR6 (được biên dịch cho SSE4) với bootstrap chuẩn (1000 bản sao bootstrap), cài đặt trong TNT phiên bản 1.1 (tháng 10 năm 2014) và PAUP* phiên bản 4.0a152 (tháng 1 năm 2017). Tất cả các phương pháp lưu một và chỉ một cây tốt nhất cho mỗi sắp hàng bootstrap. Luận án so sánh các sắp hàng DNA và protein sử dụng ma trận chi phí đều và không đều. Với dữ

liệu DNA, ma trận chi phí không đều có chi phí biến đổi cùng nhóm là 1 và chi phí biến đổi chéo nhóm là 2. Với dữ liệu protein, chi phí biến đổi giữa hai axít amin được định nghĩa là số lượng biến đổi nucleotide tối thiểu cần thiết để biến axít amin này thành axít amin kia. Ma trận chi phí này được sửa đổi để không vi phạm bất đẳng thức tam giác [92].

Chúng tôi sử dụng hai thủ tục tìm cây trong TNT, ký hiệu là tìm kiếm nhanh *fast* (thủ tục này được gợi ý bởi Pablo Goloboff - tác giả của TNT qua liên lạc cá nhân) và tìm kiếm kĩ *intensive*. fast-TNT sử dụng lệnh "*mult=rep 1 hold 1*" (nghĩa là, TNT thực hiện chiến lược xây dựng cây từng bước ngẫu nhiên, sau đó leo đồi bằng kỹ thuật TBR) để tìm kiếm cây trên sắp hàng gốc và các sắp hàng bootstrap. intensive-TNT sử dụng lệnh " *xmult = notarget hits 3 level 0 chklevel +1 1*" cho sắp hàng gốc và " *mult=rep 1 hold 1*" cho các sắp hàng bootstrap. Lệnh *xmult* kết hợp các chiến lược tìm kiếm khác nhau như ratchet, sectorial searches, tree fusing và tree drifting [31]. Vì vậy, intensive-TNT tìm kiếm không gian cây triệt để hơn fast-TNT trên sắp hàng gốc, nhưng sử dụng chiến lược tìm kiếm giống fast-TNT trên các sắp hàng bootstrap.

Chúng tôi cũng khảo sát bootstrap chuẩn cài đặt trong PAUP* bằng cách áp dụng chiến lược xây dựng cây từng bước ngẫu nhiên, sau đó leo đồi bằng TBR độc lập trên sắp hàng gốc và các sắp hàng bootstrap. Do thời gian thực hiện của PAUP* quá nhiều, chúng tôi chỉ có thể chạy PAUP* cho ma trận chi phí đều. Các lệnh TNT và PAUP* chi tiết được trình bày trong Phụ lục 2.

3.4.1 Dữ liệu mô phỏng

Để đánh giá thời gian tính toán, khả năng tìm ra cây MP và độ chuẩn xác của ước lượng bootstrap, luận án làm lại dữ liệu mô phỏng các sắp hàng DNA và protein được mô tả trong [56]. Chúng tôi đã tải xuống các sắp hàng từ cơ sở dữ liệu PANDIT [93], chọn mô hình ML phù hợp nhất và xây dựng cây ML cho mỗi sắp hàng. Những cây ML này sau đó được coi là cây đúng để sinh sắp hàng mô phỏng theo các tham

số mô hình phù hợp nhất, cùng chiều dài và khoảng trống tương tự như các sắp hàng PANDIT gốc. Cần lưu ý rằng phân tích theo tiêu chuẩn MP vi phạm các giả thiết của các mô hình phù hợp nhất được lựa chọn. Chúng tôi đã loại trừ 15 sắp hàng DNA và 17 sắp hàng protein có phân tích TNT hoặc PAUP* không hoàn thành. Do đó, bộ dữ liệu mô phỏng bao gồm 6207 sắp hàng DNA (ký hiệu DNA-PANDIT) và 6165 sắp hàng protein (ký hiệu AA-PANDIT) với thông số tóm tắt trong Bảng 3.1.

Bảng 3.1. Thông tin bộ dữ liệu mô phỏng PANDIT (loại trừ các sắp hàng có phân tích TNT hoặc PAUP* không hoàn thành).

Thông số	**Loại dữ liệu**	
	DNA	**protein**
Số sắp hàng	6207	6165
Số taxa	4-403 (trung vị: 10)	4-374 (trung vị: 10)
Số vị trí sắp hàng	24-6891 (trung vị: 567)	12-2297 (trung vị: 193)

3.4.2 Dữ liệu thực

Để đánh giá MPBoot, luận án phân tích lại 115 sắp hàng TreeBASE - bộ dữ liệu dùng để đánh giá thời gian tính toán của UFBoot2 trong phần 2.5.1, bao gồm 70 sắp hàng DNA và 45 sắp hàng protein (thông số tóm tắt trong Bảng 2.1). Tuy nhiên, chúng tôi đã phải loại trừ M9915 vì intensive-TNT đã không hội tụ. Tất cả các số liệu thống kê tổng hợp do đó dựa trên 114 sắp hàng còn lại.

3.5 Kết quả thực nghiệm

3.5.1 Thời gian tính toán

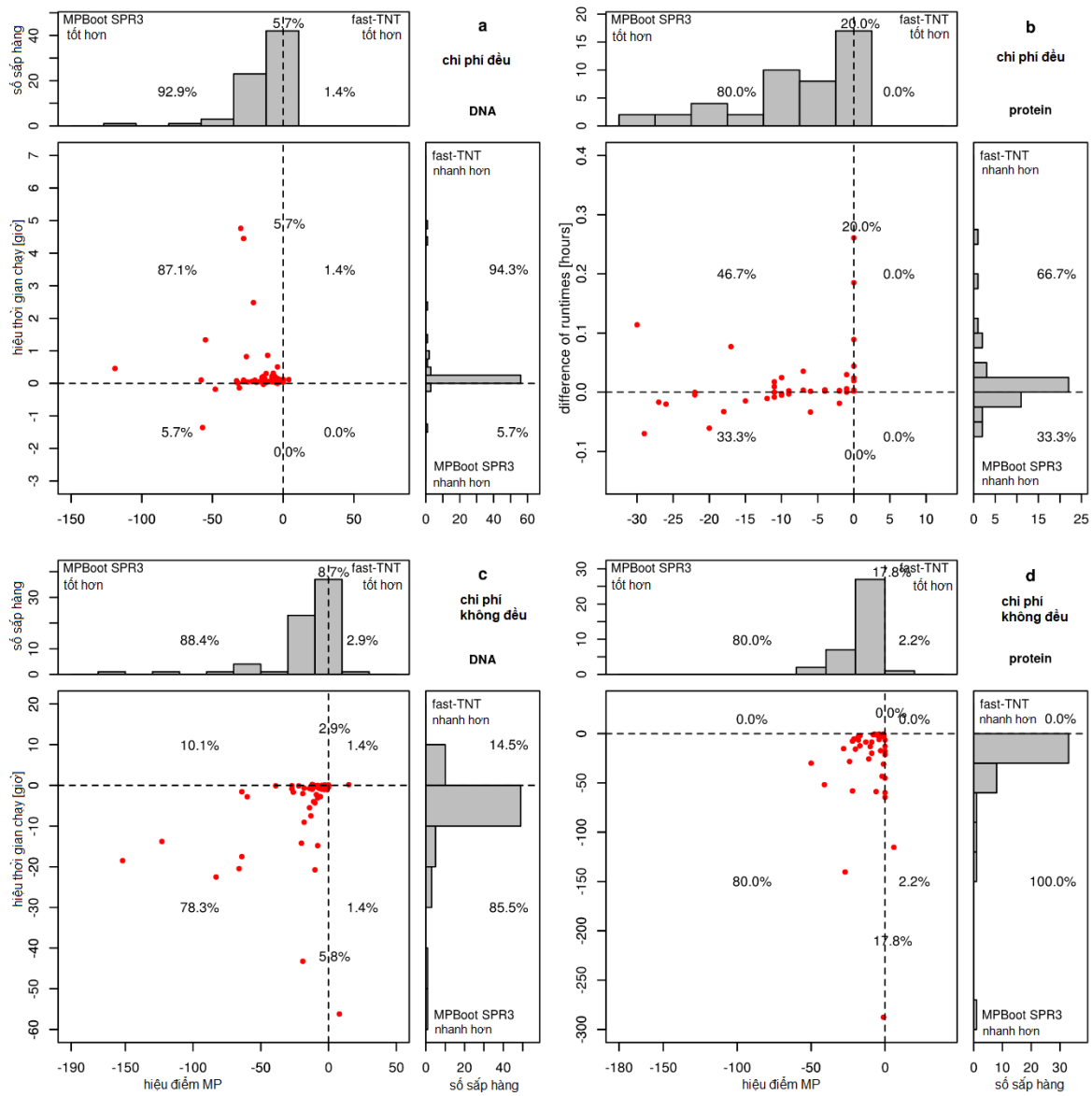
Bảng 3.2. Tổng thời gian chạy (giờ) của 5 phương pháp trên 114 sắp hàng TreeBASE. Con số in đậm ứng với phương pháp nhanh nhất theo ma trận chi phí tương ứng.

	Ma trận chi phí đều	Ma trận chi phí không đều
fast-TNT	**14.9**	1784
MPBoot SPR3	36.2	**202**
MPBoot SPR6	70.1	491
intensive-TNT	72.5	2470
PAUP*	206.1	không có số liệu

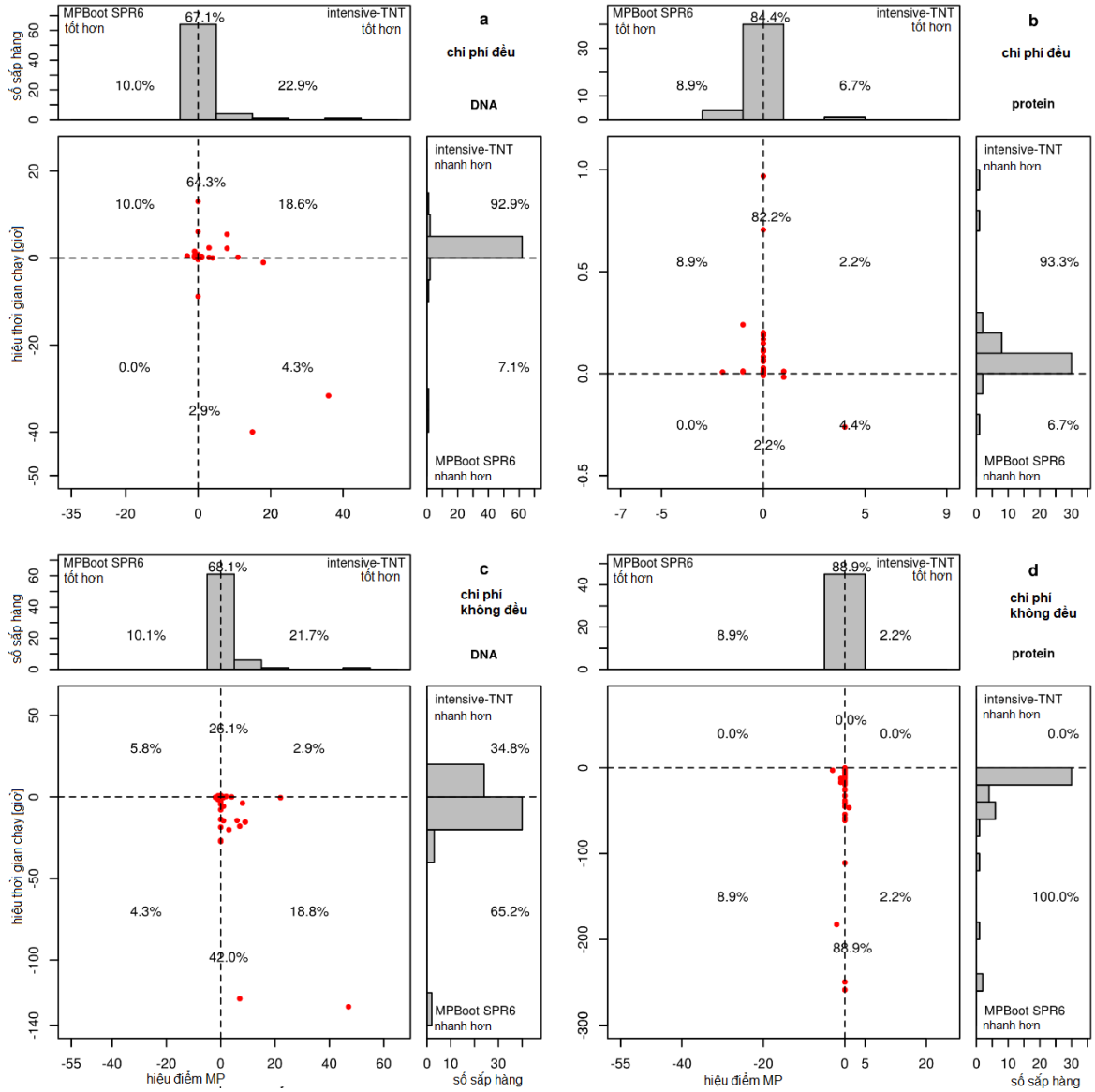
Để có xếp hạng tổng thể cho các phương pháp bootstrap khảo sát, luận án so sánh thời gian chạy tích lũy của chúng trên 114 sắp hàng TreeBASE (Bảng 3.2). Sử dụng ma trận chi phí đều, fast-TNT cần 14.9 giờ và là phương pháp nhanh nhất, tiếp theo là MPBoot SPR3 (36.2 giờ) và SPR6 (70.1 giờ). PAUP* là phương pháp chậm nhất (206.1h, chậm hơn khoảng 14 lần so với fast-TNT). Tuy nhiên, khi sử dụng ma trận chi phí không đều, MPBoot SPR3 là chương trình nhanh nhất (202 giờ), MPBoot SPR6 đứng thứ nhì (491 giờ).

Thay vì tập trung vào thời gian chạy tích lũy, ta sẽ thu được nhiều thông tin khi nghiên cứu phân bố hiệu số thời gian chạy trên tất cả các sắp hàng. Hình 3.1 cho thấy so sánh giữa fast-TNT và MPBoot SPR3. Hình 3.2 cho thấy so sánh giữa intensive-TNT và MPBoot SPR6. Trong Hình 3.1 và Hình 3.2, mỗi dấu chấm trong hình chính đại diện cho một sắp hàng gốc. Trục y cho biết hiệu số giữa thời gian CPU của hai chương trình. Trục x cho biết hiệu số giữa điểm MP của cây tốt nhất cho sắp hàng gốc được xây dựng bởi hai chương trình. Các biểu đồ phân bố tần suất ở phía trên và bên hông thể hiện các tần suất biên. Các dấu chấm bên trái của đường dọc kẻ đứt thể hiện các sắp hàng mà MPBoot tìm thấy điểm MP tốt hơn. Nếu một dấu chấm nằm dưới đường ngang kẻ đứt, phân tích bootstrap bằng MPBoot nhanh hơn. Phần trăm trong các góc phần tư của biểu đồ biểu thị tỉ lệ các sắp hàng thuộc vùng đó. Tỷ lệ phần trăm nằm ngay trên đường dọc kẻ đứt phản ánh số lượng các sắp hàng mà hai chương trình có điểm MP bằng nhau.

Theo Hình 3.1, fast-TNT nhanh hơn MPBoot SPR3 nếu dùng ma trận chi phí đều (94.3% sắp hàng DNA và 66.7% sắp hàng protein) nhưng chậm hơn nhiều so với MPBoot SPR3 nếu dùng ma trận chi phí không đều (hiệu số thời gian lên đến 55 và 280 giờ đối với DNA và protein). Ta cũng quan sát được các lợi thế tương tự về hiệu năng khi so sánh intensive-TNT và MPBoot SPR6 (Hình 3.2).



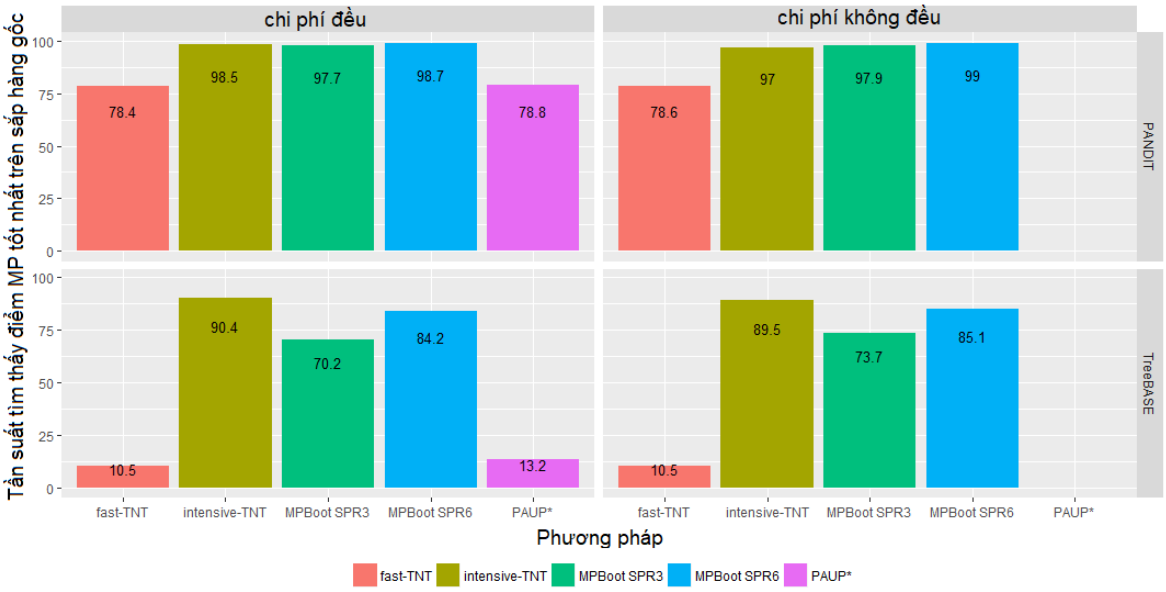
Hình 3.1. So sánh hiệu năng về thời gian chạy và điểm MP giữa MPBoot SPR3 và fast-TNT với các ma trận chi phí đều (a, b) và không đều (c, d) trên các sắp hàng DNA và axít amin thực.



Hình 3.2. So sánh hiệu năng về thời gian chạy và điểm MP giữa MPBoot SPR6 và intensive-TNT với các ma trận chi phí đều (a, b) và không đều (c, d) trên các sắp hàng DNA và axít amin thực.

3.5.2 Khả năng tìm được cây có điểm MP tốt nhất

Do tốc độ của MPBoot có thể phải đánh đổi với khả năng tìm ra điểm MP tốt nhất trên các sắp hàng gốc, luận án so sánh các điểm MP tốt nhất tìm được theo các phương pháp khác nhau cho sắp hàng gốc (cả dữ liệu mô phỏng và dữ liệu thực). Để làm điều này, chúng tôi tính tần suất mà mỗi phương pháp thu được điểm số tốt nhất trong 5 phương pháp khảo sát trên tất cả các sắp hàng (Hình 3.3).



Hình 3.3. Hiệu năng của các phương pháp khảo sát trong việc xây dựng cây MP cho sắp hàng gốc. Các biểu đồ cột cho thấy tần suất mà mỗi trong số năm phương pháp khảo sát thu được điểm MP tốt nhất cho sắp hàng gốc trong (A) bộ dữ liệu mô phỏng PANDIT và (B) bộ dữ liệu TreeBASE.

Cần lưu ý rằng, trong so sánh này, các điểm MP của cây xây dựng bởi MPBoot và TNT đã được tính lại bằng PAUP*. Lưu ý thêm rằng điểm MP tốt nhất cho một sắp hàng nhất định có thể được tìm thấy bởi hơn một phương pháp; do đó tổng các tần suất cho một bộ dữ liệu có thể lớn hơn 1. Dữ liệu của PAUP* với ma trận chi phí không đều không được trình bày do thời gian thực hiện quá nhiều.

Trên dữ liệu mô phỏng (Hình 3.3; các biểu đồ phía trên), fast-TNT và PAUP* cho thấy các tần suất tương tự nhau trong việc tìm ra điểm số tốt nhất (75% đến 82%). Điều này không gây ngạc nhiên vì chúng cài đặt các chiến lược tìm kiếm tương tự nhau. MPBoot SPR3, MPBoot SPR6 và intensive-TNT đạt được tần suất cao hơn trong việc tìm ra điểm số tốt nhất (95% đến 99.5%).

Phân tích thêm 114 sắp hàng TreeBASE cho thấy tần suất giảm đối với tất cả các phương pháp (Hình 3.3; các biểu đồ phía dưới). Đáng chú ý, tần suất đạt được điểm số tốt nhất của fast-TNT và PAUP* giảm xuống 10% -13%, trong khi MPBoot

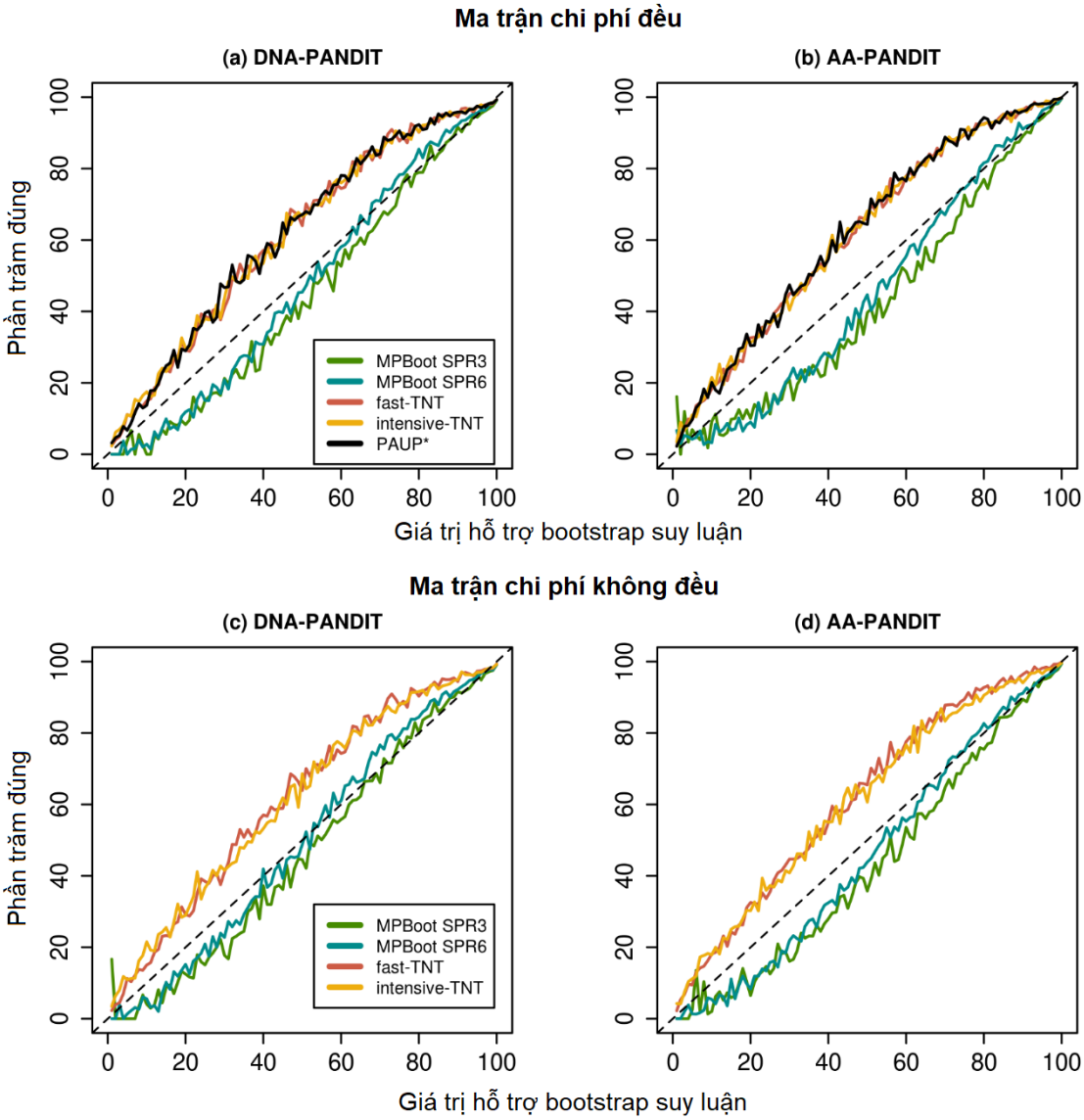
và intensive-TNT có tần suất trung bình (70% -85%) và cao (90%). Chúng tôi cũng nhận thấy rằng không có khác biệt kết quả khi sử dụng ma trận chi phí đều và không đều.

Hình 3.3 chỉ cho thấy các tần suất đạt được điểm cao nhất, để thu được nhiều thông tin hơn chúng tôi đánh giá cho từng phương pháp mức độ chênh lệch của điểm số tìm được so với điểm số tốt nhất. Để làm được điều này, chúng tôi so sánh MPBoot SPR3 và fast-TNT theo hiệu số giữa các điểm MP trên mỗi sắp hàng TreeBASE (Hình 3.1; mỗi dấu chấm ứng với một sắp hàng). Khi sử dụng ma trận chi phí đều MPBoot SPR3 có điểm số thấp hơn fast-TNT cho 92.9% sắp hàng DNA (Hình 3.1a) và 80% sắp hàng protein (Hình 3.1b). Chúng tôi quan sát được các kết quả tương tự khi sử dụng ma trận chi phí không đều (Hình 3.1c và Hình 3.1d). Tuy nhiên, intensive-TNT cho thấy điểm số tốt hơn MPBoot SPR6 trên các sắp hàng DNA (Hình 3.2a và Hình 3.2c) và kết quả tương tự trên các sắp hàng protein (Hình 3.2b và Hình 3.2d).

3.5.3 Độ chuẩn xác của ước lượng bootstrap

Luận án đã so sánh *độ chuẩn xác* của các giá trị ước lượng bootstrap gán bởi MPBoot và bootstrap chuẩn cài đặt trong fast-TNT, intensive-TNT và PAUP*. Hình 3.4a và Hình 3.4b thể hiện các hàm tính độ chuẩn xác cho năm phương pháp khảo sát khi sử dụng ma trận đều trên dữ liệu mô phỏng. Nó cho thấy rằng loại dữ liệu (nucleotides hoặc axít amin) của các sắp hàng không ảnh hưởng đến tính chuẩn xác của ước lượng bootstrap. Các phương pháp TNT và PAUP* cho ước lượng thấp hơn xác suất đúng của cạnh (Hình 3.4a và Hình 3.4b; các đường cong nằm phía trên đường chéo). Ví dụ: một cạnh có giá trị hỗ trợ PAUP* tối thiểu 80% có xác suất đúng của cạnh là 95%. Điều này khẳng định các nghiên cứu trước đó (ví dụ: [39]) rằng bootstrap chuẩn bảo thủ trong ước lượng xác suất đúng của cạnh. MPBoot SPR6 thu được các giá trị hỗ trợ bootstrap gần như không chệch, ít nhất là cho các cạnh có giá trị hỗ trợ bootstrap > 70% (Hình 3.4a và Hình 3.4b; các đường cong nằm gần đường chéo). Điều này cho phép một cách diễn giải trực quan hơn về các giá trị hỗ trợ

bootstrap. Cụ thể, để đạt được xác suất đúng của cạnh là 95%, các giá trị hỗ trợ bootstrap MPBoot SPR6 cần phải là 95%.



Hình 3.4. Độ chuẩn xác của các giá trị hỗ trợ bootstrap trên các sắp hàng DNA và protein mô phỏng PANDIT gán bởi MPBoot SPR3 (đường cong xanh lá), MPBoot SPR6 (đường cong màu xanh da trời), fast-TNT (đường cong màu đỏ), intensive-TNT (đường cong màu vàng) và PAUP* (đường cong màu đen) khi sử dụng ma trận chi phí đều (a, b) và ma trận chi phí không đều (c, d). Kích thước ngăn (bin) trên trục x là 1%.

Tương tự, chúng tôi nhận thấy rằng khi sử dụng ma trận chi phí không đều, MPBoot SPR6 ít bảo thủ hơn fast-TNT cho các giá trị hỗ trợ bootstrap> 70% (Hình 3.4c và Hình 3.4d; Luận án không khảo sát PAUP* do thời gian tính toán quá nhiều).

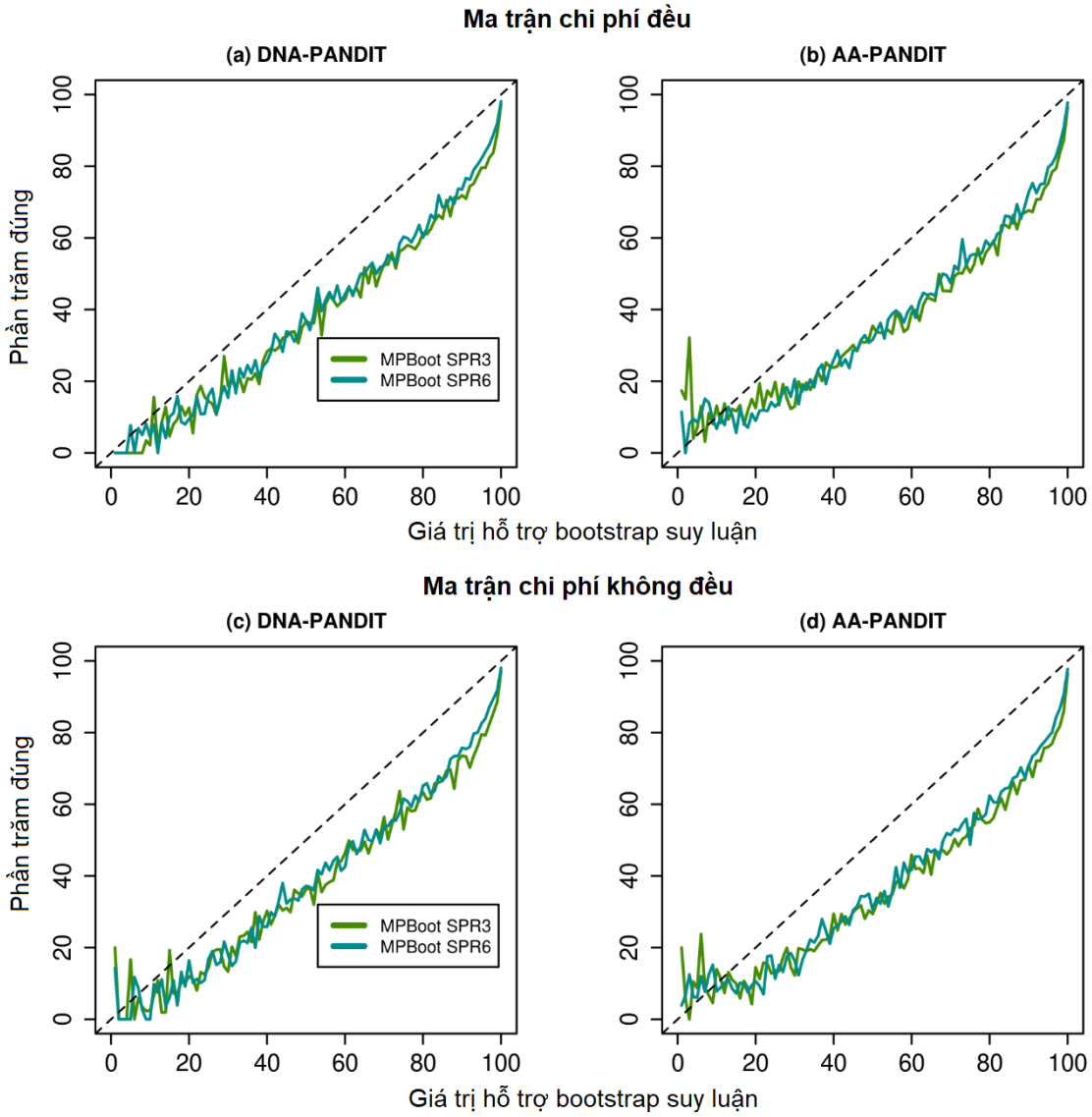
Chúng tôi kiểm tra tác động của bước tinh chỉnh tới độ chuẩn xác bằng thực nghiệm tắt bước tinh chỉnh khi dùng MPBoot phân tích dữ liệu mô phỏng PANDIT. Kết quả cho thấy trên cả 2 loại dữ liệu dù dùng sử dụng ma trận chi phí nào, giá trị hỗ trợ bootstrap tính được đều cao hơn xác suất đúng của cạnh (Hình 3.5; các đường cong võng xuống phía dưới đường chéo).

Để tiếp tục tìm hiểu sự khác biệt giữa ước lượng bootstrap của MPBoot và bootstrap chuẩn, luận án so sánh điểm MP của các cây bootstrap mà MPBoot và TNT thu được, trong trường hợp sử dụng ma trận chi phí đều. Hàm mean_score() (Hình 3.6) kí hiệu cho trung bình cộng điểm MP của các cây bootstrap thu được bởi phương pháp tương ứng. Điểm MP cho bootstrap của MPBoot SPR3 cao hơn 2.7 (trung vị; miền của hiệu điểm số: -60.8 đến 100.5) so với fast-TNT. Trong khi đó, MPBoot SPR6 đạt được điểm MP cho bootstrap thấp hơn 1 (trung vị; miền của hiệu điểm số: -63.4 đến 28.9) so với fast-TNT (Hình 3.6). Nói chung, chúng tôi đã không quan sát được khác biệt đáng kể nào giữa các điểm MP cho cây bootstrap của MPBoot và TNT.

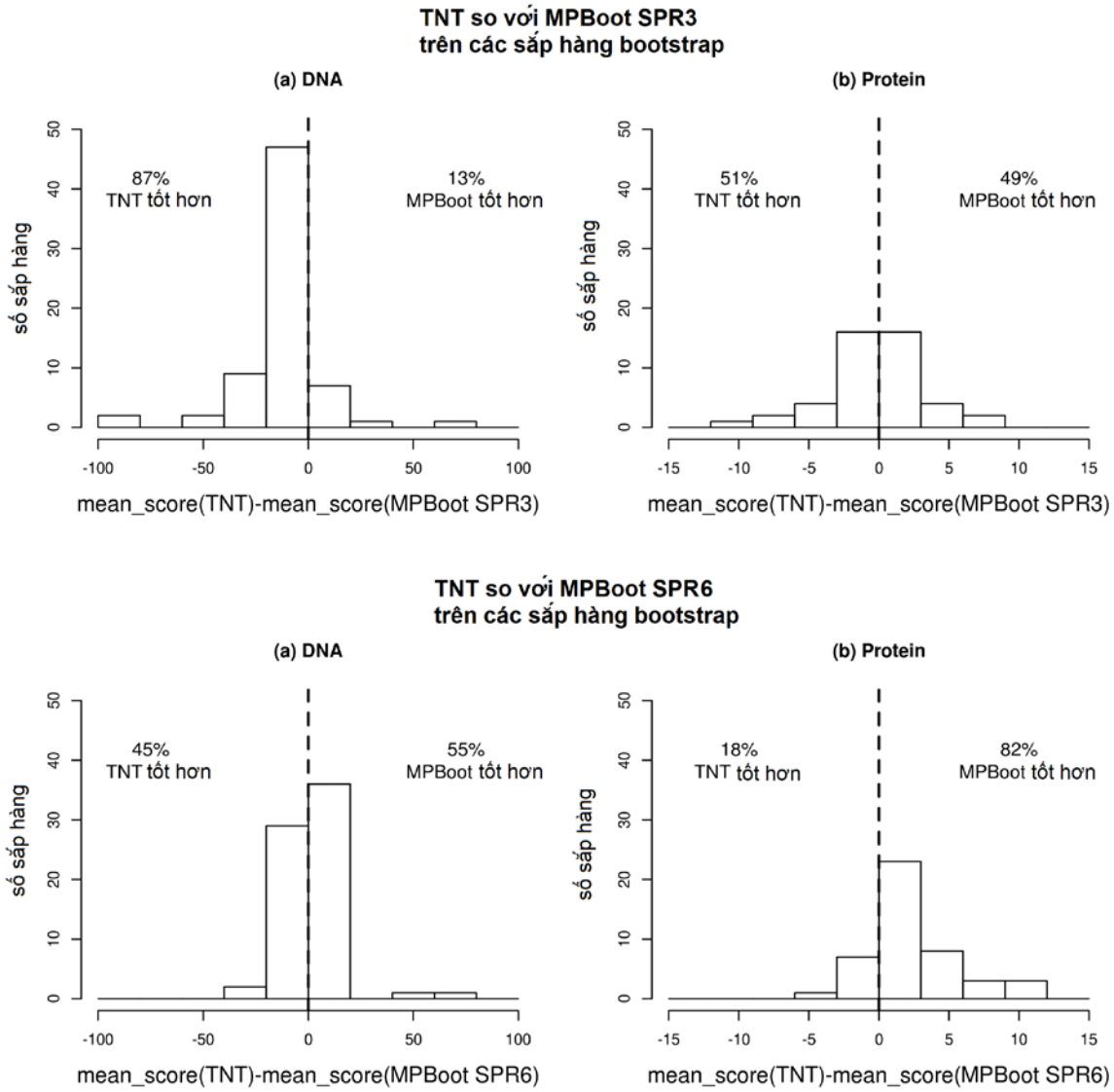
3.6 Bình luận về kết quả

Trong chương này, luận án đã trình bày về MPBoot, một thuật toán mới xây dựng cây MP, đồng thời thực hiện tìm nhanh lời giải chấp nhận được cho bootstrap theo tiêu chuẩn MP, có ý tưởng bắt nguồn từ UFBoot [56] suy luận cây theo tiêu chuẩn ML. MPBoot khác UFBoot ở 4 khía cạnh chính: (i) nó tính chính xác (thay vì tính xấp xỉ) các điểm MP cho cây bootstrap, (ii) nó sử dụng kỹ thuật SPR thay cho NNI trong phép tìm kiếm leo đồi, (iii) nó vận dụng parsimony ratchet (iv) nó áp dụng *bước tinh chỉnh* để cải thiện điểm số của những cây bootstrap tốt nhất thu được trong

bước khám phá, nhờ sử dụng trực tiếp các sắp hàng bootstrap. Luận án đã chỉ ra rằng sự kết hợp này cho phép khám phá hiệu quả không gian cây theo tiêu chuẩn MP.



Hình 3.5. Độ chuẩn xác của các giá trị hỗ trợ bootstrap trên dữ liệu mô phỏng PANDIT gán bởi MPBoot SPR3 (xanh lá), MPBoot SPR6 (xanh da trời) khi tắt bước tinh chỉnh.



Hình 3.6. Phân bố của trung bình hiệu điểm MP giữa các cây bootstrap của TNT và MPBoot SPR3 (trên) và của TNT và MPBoot SPR6 (dưới) trên các sắp hàng DNA (a) và protein (b) thực.

So với bootstrap chuẩn được cài đặt trong PAUP* và TNT, MPBoot cho hiệu năng tốt cả về độ chuẩn xác bootstrap và thời gian chạy trên các bộ dữ liệu mô phỏng cũng như dữ liệu thực. So với TNT về mặt tìm kiếm cây trên sắp hàng gốc, MPBoot đạt được điểm MP tốt hơn fast-TNT và ngang bằng intensive-TNT. Về mặt phân tích bootstrap, MPBoot thu được điểm MP trên bootstrap ngang bằng TNT.

Các nghiên cứu trước đây đã chỉ ra rằng xu hướng bảo thủ của bootstrap chuẩn là do hiện tượng *rogue taxa* [3,51,95]. Đây là những đỉnh lá (loài/taxa) có thể đặt ở các vị trí khác nhau trong cây mà không thay đổi điểm MP (do thiếu thông tin tiến hóa). Chúng tôi dự đoán rằng xu hướng ít bảo thủ của MPBoot (Hình 3.4) có thể là kết quả của việc giảm hiệu ứng rogue taxa. Cụ thể như sau: tính trung bình, một phần ba các vị trí trong sắp hàng gốc không được lấy mẫu lại trong sắp hàng bootstrap, điều này làm tăng hiệu ứng rogue taxa khi phép tìm kiếm cây được thực hiện độc lập cho mỗi sắp hàng bootstrap như trong bootstrap chuẩn. Ngược với điều này, MPBoot thực hiện việc tìm kiếm cây chỉ trên sắp hàng gốc (với một bước tinh chỉnh nhỏ ở cuối), sẽ làm giảm tác động của rogue taxa vì điểm MP được tính theo các vị trí của sắp hàng gốc. Đồng thời, MPBoot vẫn thu được những cây bootstrap với điểm MP ngang bằng TNT. Tìm hiểu so sánh các cấu trúc của cây bootstrap do MPBoot và TNT thu được để xác định các rogue taxa và từ đó giải thích xu hướng không chệch của MPBoot sẽ rất thú vị, nhưng ngoài phạm vi của luận án này.

Ta dễ dàng nhận thấy đánh đổi giữa điểm MP và thời gian chạy khi so sánh giữa fast-TNT và intensive-TNT hoặc giữa MPBoot SPR3 và MPBoot SPR6. Lưu ý rằng, MPBoot SPR6 đôi khi không tìm thấy cây tốt hơn MPBoot SPR3 mặc dù nó tìm kiếm kỹ hơn. Tính chất ngẫu nhiên của thuật toán tìm kiếm cây khiến MPBoot SPR6 không thể đảm bảo chắc chắn sẽ luôn tốt hơn. Mặc dù fast-TNT và PAUP* sử dụng cùng một chiến lược tìm kiếm cây, fast-TNT nhanh hơn PAUP* đáng kể do mã nguồn trong TNT được tối ưu rất tốt. MPBoot có thể sẽ nhanh hơn nữa nếu được cài đặt thêm các kỹ thuật tối ưu mã nguồn và các kỹ thuật tính toán MP hiệu quả như đề cập trong [29]. MPBoot chậm hơn TNT khi sử dụng ma trận chi phí đều nhưng nhanh hơn khi sử dụng ma trận chi phí không đều. Lợi thế này của MPBoot so với TNT là do các kỹ thuật tính toán MP hiệu quả trong TNT phụ thuộc nhiều vào các phép toán trên bit. Tuy nhiên, phần lớn các phép toán này không thể áp dụng cho ma trận chi phí không đều. Bên cạnh đó, khi dùng ma trận chi phí không đều, việc tính toán điểm MP cho một cây trên một sắp hàng bootstrap trong quá trình tìm kiếm cây bootstrap

tương ứng (như trong bất cứ cài đặt bootstrap chuẩn nào) là tốn kém hơn nhiều so với việc áp dụng (3.2) (như trong cài đặt của MPBoot).

Luận án cũng so sánh thời gian tính toán giữa MPBoot và phương pháp xỉ nhanh bootstrap theo tiêu chuẩn ML (UFBoot2). MPBoot nhanh hơn UFBoot2 khoảng 1.2 và 8.6 lần tương ứng trên các sắp hàng DNA và protein.

Với các giá trị hỗ trợ bootstrap> 70%, MPBoot SPR3 cho ước lượng cao hơn một chút so với xác suất đúng của cạnh trên sắp hàng protein nhưng khá chuẩn xác trên sắp hàng DNA (Hình 3.4; các đường cong màu xanh lá). MPBoot SPR6 cho ước lượng bootstrap gần như không chệch trên các sắp hàng protein nhưng cho ước lượng thấp hơn một chút so với xác suất đúng của cạnh trên các sắp hàng DNA với các giá trị hỗ trợ bootstrap> 70% (Hình 3.4; đường cong màu xanh da trời). Do đó, để có được cách diễn giải trực quan về các giá trị hỗ trợ bootstrap, người dùng nên sử dụng MPBoot SPR6. Cần lưu ý rằng, các giá trị hỗ trợ bootstrap thu được bởi MPBoot có khuynh hướng cao hơn so với PAUP*, fast-TNT và intensive-TNT; do đó chúng không so sánh trực tiếp được. Ví dụ, một cạnh có giá trị hỗ trợ theo fast-TNT là 70-80% có thể có giá trị hỗ trợ theo MPBoot là 95%, tương đương với khả năng 95% là cạnh đúng.

MPBoot xây dựng tập xấp xỉ cho tập hợp cây bootstrap chuẩn đồng thời với việc lấy mẫu cây cho sắp hàng gốc. Mỗi cây được duyệt trong quá trình xây dựng cây trên sắp hàng gốc sẽ được tính điểm MP ngay lập tức trên tất cả các sắp hàng bootstrap. Điều kiện dừng có mục đích xác định một số lượng hợp lý các phép leo đồi để tìm ra cây tốt nhất cho sắp hàng gốc. Do đó, ngay cả khi việc tìm kiếm kết thúc sớm, tất cả các sắp hàng bootstrap đều được xem xét.

MPBoot cung cấp tùy chọn để giữ nhiều cây tốt nhất ngang nhau cho mỗi sắp hàng bootstrap. Sử dụng tùy chọn này, chúng tôi đã quan sát thấy rằng, đối với mỗi sắp hàng TreeBASE, 85% các bản sao bootstrap có tập các cây tốt nhất ngang nhau thu được trước bước tinh chỉnh, được tối ưu thành cùng một cây bootstrap sau khi

tinh chỉnh. Vì thuật toán chỉ thu thập cây bootstrap sau bước tinh chỉnh, tùy chọn này không làm thay đổi độ chuẩn xác của MPBoot nhưng lại dẫn tới chi phí tính toán cao hơn. Vì vậy, MPBoot mặc định chỉ giữ duy nhất một cây bootstrap tốt nhất cho mỗi sắp hàng bootstrap. Hơn nữa, bước tinh chỉnh là cần thiết. Nếu không có bước tinh chỉnh, MPBoot có xu hướng cho ước lượng cao hơn xác suất đúng của cạnh (Hình 3.5).

Trong luận án này, tất cả các phương pháp bootstrap khảo sát đều giữ một cây duy nhất cho mỗi sắp hàng bootstrap và sử dụng cùng một phương pháp (tính tần suất cạnh) để tóm tắt các cây bootstrap. Tuy nhiên, cần lưu ý rằng, MPBoot/PAUP* và TNT sử dụng các cách tiếp cận khác nhau để tóm tắt các cây bootstrap nếu được cấu hình để giữ nhiều cây tốt nhất ngang nhau cho mỗi sắp hàng bootstrap. MPBoot và PAUP* trước tiên gán cho mỗi cây bootstrap một trọng số bằng nghịch đảo số lượng cây trên mỗi bản sao bootstrap tương ứng. Sau đó, chúng tính giá trị hỗ trợ bootstrap. TNT trước tiên tính toán một cây đồng thuận chặt (strict consensus) cho tập các cây bootstrap của từng bản sao bootstrap. Sau đó, nó tính giá trị hỗ trợ bootstrap từ những cây đồng thuận chặt này [30]. Ngoài các phương pháp trình bày ở đây, còn các tiếp cận khác để tóm tắt các cây bootstrap, ví dụ như phương pháp GC [30].

Thời gian chạy của tất cả các phương pháp bootstrap chủ yếu được xác định bởi số lượng bản sao bootstrap. Intensive-TNT sẽ nhanh hơn MPBoot khi sử dụng số lượng bản sao bootstrap ít đi. Ví dụ, intensive-TNT và MPBoot trở nên nhanh hơn tương ứng là 7.3 và 1.8 lần, khi thực hiện với 100, thay vì 1000 bản sao bootstrap, trên các bộ dữ liệu thực, sử dụng ma trận chi phí đều. Mặc dù các nhà sinh vật có thể chỉ thực hiện phân tích bootstrap với 100 bản sao nhằm rút ngắn thời gian, nhưng các nghiên cứu lý thuyết khuyến nghị sử dụng hàng ngàn bản sao bootstrap để đạt được kết quả có độ tin cậy cao [37]. Pattengale và cộng sự [65] khẳng định rằng số lượng tối thiểu bản sao bootstrap lại phụ thuộc rất nhiều vào dữ liệu. Dựa trên các phân tích dữ liệu thực, họ kết luận rằng công thức Hedges [37] cung cấp một cận trên hợp lý trên cho số lượng bản sao bootstrap.

Chúng tôi cũng khảo sát hiệu năng của MPBoot với bán kính SPR lớn hơn 6 và thấy rằng tăng bán kính có thể tìm được cây có điểm số tốt hơn một chút nhưng chi phí tính toán cao hơn. Hơn nữa, MPBoot SPR6 đã thu được cây bootstrap với điểm MP tương đương, thậm chí còn tốt hơn TNT - với chiến lược tìm kiếm dựa trên TBR. Điều này cho thấy MPBoot SPR6 thực hiện tốt việc tìm kiếm cả trên sắp hàng gốc và cả trên các sắp hàng bootstrap. Nixon [61] đã nhận thấy rằng tỉ lệ phần trăm ratchet từ 5% đến 25% là hiệu quả trong phân tích của ông về điểm MP trên sắp hàng gốc. Tuy nhiên, phân tích của chúng tôi trên dữ liệu thực cho thấy tỷ lệ phần trăm ratchet bằng 50% cho điểm MP tốt nhất. Do đó, MPBoot mặc định đặt bán kính SPR là 6 và tỷ lệ ratchet là 50%. Tuy vậy, người dùng có thể tùy chỉnh các tham số này nếu cần. Chúng tôi sử dụng bách phân vị thứ 90 trong việc xác định MP_{max} sau khi đã khảo sát độ chuẩn xác và thời gian chạy của các bách phân vị thấp hơn và cao hơn. Trên dữ liệu mô phỏng, bách phân vị thứ 90 cho sự cân bằng tốt nhất cho cả hai.

Trong tương lai, chúng tôi dự tính mở rộng khả năng linh hoạt của MPBoot trong tìm kiếm bằng cách tích hợp kỹ thuật xáo trộn cây TBR và phát triển lược đồ tương tự tìm nhanh lời giải chấp nhận được cho jackknife theo tiêu chuẩn MP [18]. Một khả năng mở rộng khác trong tương lai là song song hóa MPBoot. Ở đây, tính toán REPS có thể được thực hiện cho mỗi sắp hàng bootstrap một cách độc lập, và do đó có thể đồng thời. Đối với việc tìm kiếm cây trên sắp hàng gốc, ta có thể song song hóa việc tính toán điểm MP cho các vị trí trên sắp hàng, sử dụng lược đồ bộ nhớ chia sẻ hoặc phân phối các vòng lặp tìm kiếm độc lập riêng biệt cho các CPU khác nhau [57].

3.7 Kết luận chương

Chương này trình bày về MPBoot, một phương pháp mới để tìm kiếm hiệu quả cây MP, đồng thời tìm kiếm hiệu quả tập hợp xấp xỉ kết quả bootstrap chuẩn theo tiêu chuẩn MP. Luận án đã so sánh MPBoot với bootstrap chuẩn cài đặt trong TNT và PAUP* sử dụng các ma trận chi phí khác nhau. MPBoot cho điểm MP tốt hơn fast-

TNT và PAUP* và tương đương với intensive-TNT. MPBoot SPR6 thu được các giá trị hỗ trợ bootstrap gần như không chệch bất kể loại dữ liệu sử dụng và ma trận chi phí cụ thể. MPBoot có thời gian chạy ít hơn đáng kể so với PAUP*. MPBoot được chúng tôi cài đặt trong phần mềm cùng tên. Đây là phần mềm dễ sử dụng, miễn phí. Chúng tôi công khai mã nguồn mở (theo GNU General Public License) của MPBoot trên địa chỉ http://www.iqtree.org/mpboot với các chương trình chạy biên dịch sẵn cho Mac OSX và Linux.

Các kết quả nghiên cứu của chương này đã được công bố một phần trong một bài *kỉ yếu Hội nghị Quốc tế về Kỹ nghệ tri thức và Hệ thống lần thứ 8 (KSE 2016)* (công trình khoa học số 1) và trong một bài báo đăng trên tạp chí quốc tế *BMC Evolutionary Biology* năm 2018 (công trình khoa học số 3).

KẾT LUẬN

Xây dựng cây tiến hóa dựa trên dữ liệu sinh học phân tử và đánh giá độ hỗ trợ thống kê cho cây dựng được là hai bước quan trọng trong quy trình phân tích tiến hóa hiện đại. Luận án tập trung vào bài toán xây dựng cây bootstrap tiến hóa theo tiêu chuẩn ML và MP vì đây là hai tiếp cận được sử dụng rộng rãi trong phân tích tiến hóa. Phương pháp SBS có hai hạn chế chính là (i) tốn kém về thời gian thực hiện và (ii) cho ước lượng bootstrap thấp hơn xác suất đúng của cạnh. Sự gia tăng không ngừng về quy mô dữ liệu sinh học phân tử phục vụ phân tích tiến hóa đặt ra đòi hỏi phải khắc phục các hạn chế này. Luận án có 4 đóng góp chính như sau:

Một là, đề xuất phương pháp UFBoot2 giải nhanh bài toán xây dựng cây bootstrap tiến hóa theo tiêu chuẩn ML với bốn cải tiến quan trọng so với phương pháp UFBoot: (i) Cải tiến tốc độ với đề xuất thuật toán pruning nhanh và các kỹ thuật tối ưu mã nguồn, nhờ đó nhanh hơn phiên bản cũ trung bình 2.4 lần; (ii) Cải tiến để xử lý các đỉnh đa phân tốt hơn, nhờ đó thể hiện thành công trên thực nghiệm mô phỏng từ cây hình sao; (iii) Cải tiến để giảm ảnh hưởng của vi phạm mô hình, nhờ đó cho ước lượng sát với xác suất đúng của cạnh kể cả khi vi phạm mô hình nhiều và (iv) Cải tiến mở rộng để phân tích sắp hàng nhiều gen.

Hai là, phỏng theo ý tưởng của UFBoot cho ML, luận án đã đề xuất một phương pháp MPBoot mới để tìm kiếm hiệu quả cây MP, đồng thời tìm kiếm hiệu quả lời giải chấp nhận được cho bài toán xây dựng cây bootstrap tiến hóa theo tiêu chuẩn MP. MPBoot có thể sử dụng ma trận chi phí đều hoặc không đều. Luận án đã thực hiện các thực nghiệm trên cả dữ liệu mô phỏng và các bộ dữ liệu sinh học lớn để so sánh MPBoot và các phương pháp cài đặt trong hai công cụ phân tích theo tiêu chuẩn MP tốt nhất hiện nay là TNT và PAUP* trên các mặt: thời gian tính toán, khả năng tìm ra cây có điểm MP tốt nhất và độ chuẩn xác của ước lượng bootstrap. Bootstrap nhanh của MPBoot thực hiện nhanh hơn bootstrap chuẩn cài đặt trong PAUP* khi sử dụng ma trận chi phí đều; nhanh hơn fast-TNT khi sử dụng ma trận chi phí không đều. MPBoot xây dựng được cây MP có điểm MP tương đương với intensive-TNT. Trong

khi các bản cài đặt bootstrap chuẩn đều cho ước lượng thấp về xác suất đúng của cạnh, MPBoot SPR6 thu được các giá trị hỗ trợ bootstrap gần như không chệch bất kể loại dữ liệu sử dụng và ma trận chi phí cụ thể. MPBoot, do đó, có thể thay thế hiệu quả các tiếp cận bootstrap chuẩn theo tiêu chuẩn MP.

Ba là, phương pháp UFBoot2 đề xuất trong luận án đã được chúng tôi tích hợp vào hệ thống IQ-TREE (địa chỉ website: http://www.iqtree.org) được Zhou và cộng sự đánh giá là phần mềm mã nguồn mở tốt nhất hiện nay cho phân tích cây tiến hóa theo tiêu chuẩn ML [104] và có nhiều người sử dụng.

Bốn là, các đề xuất về phương pháp mới MPBoot đã được chúng tôi cài đặt trong phần mềm mã nguồn mở MPBoot (địa chỉ website: http://www.iqtree.org/mpboot).

Luận án, tuy vậy, còn hạn chế trong giải thích xu hướng ước lượng chệch/không chệch của từng phương pháp trên cơ sở toán học và thống kê; trong mở rộng khảo sát các mức độ vi phạm mô hình khác nhau trong nhiều cấu hình mô phỏng khác nhau cho UFBoot2; hay trong mở rộng so sánh với các phương pháp phân tích tiến hóa sử dụng Bayes (như MrBayes [70], BEAST [14]).

Bên cạnh nghiên cứu khắc phục các hạn chế này, chúng tôi có một số đề xuất để phát triển kết quả nghiên cứu của luận án như sau: *Thứ nhất*, nghiên cứu thiết kế so sánh với các phương pháp hiện có, kết hợp khảo sát người dùng để nhận định về tính ổn định của từng phương pháp. *Thứ hai*, nghiên cứu cải thiện các phương pháp để cài đặt trên hệ thống GPU. *Thứ ba*, nghiên cứu kết hợp thế mạnh về tốc độ của UFBoot2 với thế mạnh về độ chuẩn xác bootstrap cho dữ liệu lớn của phương pháp BOOSTER [51]. BOOSTER, được công bố sau UFBoot2, là một công thức mới để tính giá trị hỗ trợ cho một cạnh từ tập cây bootstrap của SBS. Nó nhằm khắc phục điểm yếu của công thức truyền thống dùng tỉ lệ cây bootstrap có chứa cạnh đó. Trên dữ liệu nhiều taxa, SBS kết hợp công thức truyền thống có xu hướng tính ra giá trị hỗ trợ rất thấp cho các cạnh nằm sâu trong cây. Do tập cây bootstrap của UFBoot2 và

SBS khác nhau, để kết hợp UFBoot2 và BOOSTER có thể cần hiệu chỉnh từ cả 2 phía, do đó cần có nghiên cứu sâu hơn. *Thứ ba*, nghiên cứu phát triển một thước đo toàn diện hơn về độ hỗ trợ thống kê khi phân tích bootstrap dữ liệu nhiều gen. Với dữ liệu nhiều gen, ta có 1 cây tiến hóa cho mỗi gen và 1 cây tiến hóa loài (dựa trên kết hợp tất cả các cây tiến hóa gen). UFBoot2 đã có đề xuất sinh sắp hàng bootstrap cho dữ liệu nhiều gen, các giá trị hỗ trợ bootstrap tính cho cây tiến hóa loài thể hiện mức độ hỗ trợ từ dữ liệu. Để tính tới cây tiến hóa gen, nó cần kết hợp với ý tưởng mới đây của Minh và cộng sự [55] về thước đo gCF tính tỉ lệ xuất hiện trên các cây tiến hóa gen cho mỗi cạnh trên cây tiến hóa loài.

DANH MỤC CÁC CÔNG TRÌNH KHOA HỌC CỦA TÁC GIẢ LIÊN QUAN ĐẾN LUẬN ÁN

1. **Hoang DT**, Le SV, Flouri T, Stamatakis A, von Haeseler A, Minh BQ (2016), "A new phylogenetic tree sampling method for maximum parsimony bootstrapping and proof-of-concept implementation," In: 2016 Eighth International Conference on Knowledge and Systems Engineering (KSE), pp.1–6. Giải nhì bài báo xuất sắc cho học viên (Runner up Best Student Paper).
2. **Hoang DT**, Chernomor O, von Haeseler A, Minh BQ, Le SV (2018), "UFBoot2: Improving the ultrafast bootstrap approximation," *Molecular Biology and Evolution*, Vol. 35(2), pp.518–522. Tạp chí thuộc danh mục cơ sở dữ liệu của ISI. Được chọn vào tuyển tập MBE Citation Classics (2019 Edition). Giải Nhất - Giải thưởng Công trình, Sản phẩm Khoa học Công nghệ Xuất sắc 2018 của Trường Đại học Công Nghệ.
3. **Hoang DT**, Le SV, Flouri T, Stamatakis A, von Haeseler A, Minh BQ (2018 Feb), "MPBoot: fast phylogenetic maximum parsimony tree inference and bootstrap approximation," *BMC Evolutionary Biology*, Vol. 18(1), p.11. Tạp chí thuộc danh mục cơ sở dữ liệu của ISI. (Tiêu đề của bản nộp đầu tiên: 'MPBoot: Novel and fast approximation for maximum parsimony bootstrap')

Danh mục này gồm 03 công trình.

TÀI LIỆU THAM KHẢO

Tiếng Việt

1. Phạm Thị Trân Châu, Trần Thị Áng (2006), *Hóa sinh học*, Nhà xuất bản Giáo dục.
2. Lê Đức Trình (2001), *Sinh học phân tử của tế bào*, Nhà xuất bản Khoa học và Kỹ thuật.

Tiếng Anh

3. Aberer AJ, Krompass D, Stamatakis A (2013), "Pruning rogue taxa improves phylogenetic accuracy: An efficient algorithm and webservice," *Systematic Biology*, Vol. 62(1), pp.162–166.
4. Adachi J, Hasegawa M (1996), *MOLPHY version 2.3: programs for molecular phylogenetics based on maximum likelihood*, Institute of Statistical Mathematics Tokyo.
5. Anisimova M, Gascuel O (2006), "Approximate likelihood-ratio test for branches: A fast, accurate, and powerful alternative," *Systematic Biology*, Vol. 55(4), pp.539–552.
6. Anisimova M, Gil M, Dufayard J-F, Dessimoz C, Gascuel O (2011), "Survey of branch support methods demonstrates accuracy, power, and robustness of fast likelihood-based approximation schemes," *Systematic Biology*, Vol. 60(5), pp.685–699.
7. Bouchenak-Khelladi Y, Salamin N, Savolainen V, Forest F, van der Bank M, Chase MW, et al. (2008), "Large multi-gene phylogenetic trees of the grasses (Poaceae): Progress towards complete tribal and generic level sampling," *Molecular Phylogenetics and Evolution*, Vol. 47(2), pp.488–505.
8. Bush RM (2001 May 1), "Predicting adaptive evolution," *Nature Reviews Genetics*, Vol. 2, pp.387–392.

9. Chernomor O, von Haeseler A, Minh BQ (2016), “Terrace aware data structure for phylogenomic inference from supermatrices,” *Systematic Biology*, Vol. 65(6), pp.997–1008.

10. Chor B, Tuller T (2005 Jun), “Maximum likelihood of evolutionary trees: hardness and approximation,” *Bioinformatics*, Vol. 21(Suppl 1), pp.i97–i106.

11. Creighton TE (1993), *Proteins : structures and molecular properties*, 2nd ed., New York: W. H. Freeman.

12. Dell’Ampio E, Meusemann K, Szucsich NU, Peters RS, Meyer B, Borner J, et al. (2014), “Decisive data sets in phylogenomics: lessons from studies on the phylogenetic relationships of primarily wingless insects,” *Molecular Biology and Evolution*, Vol. 31(1), pp.239–249.

13. Douady CJ, Delsuc F, Boucher Y, Doolittle WF, Douzery EJP (2003), “Comparison of Bayesian and maximum likelihood bootstrap measures of phylogenetic reliability,” *Molecular Biology and Evolution*, Vol. 20(2), pp.248–254.

14. Drummond AJ, Suchard MA, Xie D, Rambaut A (2012), “Bayesian phylogenetics with BEAUti and the BEAST 1.7,” *Molecular Biology and Evolution*, Vol. 29(8), pp.1969–1973.

15. Efron B (1979), “Bootstrap methods: another look at the jackknife,” *The Annals of Statistics*, Vol. 7(1), pp.1–26.

16. Efron B, Gong G (1983), “A leisurely look at the bootstrap, the jackknife, and cross-validation,” *The American Statistician*, Vol. 37(1), pp.36–48.

17. Fabre P-. H, Rodrigues A, Douzery EJP (2009), “Patterns of macroevolution among Primates inferred from a supermatrix of mitochondrial and nuclear DNA,” *Molecular Phylogenetics and Evolution*, Vol. 53(3), pp.808–825.

18. Farris JS, Albert VA, Källersjö M, Lipscomb D, Kluge AG (1996), “Parsimony

jackknifing outperforms neighbor-joining," *Cladistics*, Vol. 12(2), pp.99–124.

19. Felsenstein J (1973), "Maximum likelihood and minimum-steps methods for estimating evolutionary trees from data on discrete characters," *Systematic Biology*, Vol. 22(3), pp.240–249.

20. Felsenstein J (1981), "Evolutionary trees from DNA sequences: a maximum likelihood approach," *Journal of Molecular Evolution*, Vol. 17(6), pp.368–376.

21. Felsenstein J (1985), "Confidence limits on phylogenies : an approach using the bootstrap," *Evolution*, Vol. 39(4), pp.783–791.

22. Felsenstein J (2004), *Inferring phylogenies*, 2nd ed., Sunderland (MA): Sinauer Associates, Inc.

23. Fitch WM (1971), "Toward defining the course of evolution: minimum change for a specific tree topology," *Systematic Zoology*, Vol. 20(4), pp.406–416.

24. Flouri T, Izquierdo-Carrasco F, Darriba D, Aberer AJ, Nguyen L-T, Minh BQ, et al. (2015), "The phylogenetic likelihood library," *Systematic Biology*, Vol. 64(2), pp.356–362.

25. Gadagkar SR, Rosenberg MS, Kumar S (2005), "Inferring species phylogenies from multiple genes: Concatenated sequence tree versus consensus gene tree," *Journal of Experimental Zoology Part B: Molecular and Developmental Evolution*, Vol. 304B(1), pp.64–74.

26. Galtier N, Gascuel O, Jean-Marie A (2005), "Markov models in molecular evolution," In: Statistical Methods in Molecular Evolution, New York, NY: Springer New York, pp.3–24.

27. Gascuel O (1997 Jul), "BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data," *Molecular biology and evolution*, Vol. 14(7), pp.685–695.

28. Goldman N (1993), "Statistical tests of models of DNA substitution," *Journal*

of Molecular Evolution, Vol. 36(2), pp.182–198.

29. Goloboff PA (1996), "Methods for faster parsimony analysis," *Cladistics*, Vol. 12(3), pp.199–220.

30. Goloboff PA, Farris JS, Källersjö M, Oxelman B, Ramírez MJ, Szumik CA (2003), "Improvements to resampling measures of group support," *Cladistics*, Vol. 19(4), pp.324–332.

31. Goloboff PA, Farris JS, Nixon KC (2008), "TNT, a free program for phylogenetic analysis," *Cladistics*, Vol. 24(5), pp.774–786.

32. Graham RL, Foulds LR (1982), "Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time," *Mathematical Biosciences*, Vol. 60(2), pp.133–142.

33. Guindon S, Dufayard J-F, Lefort V, Anisimova M, Hordijk W, Gascuel O (2010), "New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of PhyML 3.0," *Systematic Biology*, Vol. 59(3), pp.307–321.

34. Guindon S, Gascuel O (2003), "A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood," *Systematic Biology*, Vol. 52(5), pp.696–704.

35. Hartigan JA (1973), "Minimum mutation fits to a given tree," *Biometrics*, Vol. 29(1), pp.53–65.

36. Hasegawa M, Kishino H (1994), "Accuracies of the simple methods for estimating the bootstrap probability of a maximum-likelihood tree," *Molecular Biology and Evolution*, Vol. 11(1), p.142.

37. Hedges SB (1992 Mar), "The number of replications needed for accurate estimation of the bootstrap P value in phylogenetic studies," *Molecular biology and evolution*, Vol. 9(2), pp.366–369.

38. Herron JC, Freeman S (2014), *Evolutionary analysis*, 5th ed., Pearson.

39. Hillis DM, Bull JJ (1993), "An empirical test of bootstrapping as a method for assessing confidence in phylogenetic analysis," *Systematic Biology*, Vol. 42(2), pp.182–192.

40. Hinchliff CE, Roalson EH (2013), "Using supermatrices for phylogenetic inquiry: an example using the sedges," *Systematic Biology*, Vol. 62(2), pp.205–219.

41. Holland JH (1975), *Adaptation in natural and artificial systems*, University of Michigan Press.

42. Jia F, Lo N, Ho SYW (2014), "The impact of modelling rate heterogeneity among sites on phylogenetic estimates of intraspecific evolutionary rates and timescales," *PLOS ONE*, Vol. 9(5), pp.1–8.

43. Jukes TH, Cantor CR (1969), "Evolution of protein molecules," In: Munro H, editor. Mammalian protein metabolism, New York: Academic Press, pp.21–132.

44. Karpiński P, McDonald J (2017), "A high-performance portable abstract interface for explicit SIMD vectorization," In: Proceedings of the 8th International Workshop on Programming Models and Applications for Multicores and Manycores, New York, NY, USA: ACM, pp.21–28.

45. Kishino H, Miyata T, Hasegawa M (1990), "Maximum likelihood inference of protein phylogeny and the origin of chloroplasts," *Journal of Molecular Evolution*, Vol. 31(2), pp.151–160.

46. Kozlov AM, Darriba D, Flouri T, Morel B, Stamatakis A (2019), "RAxML-NG: A fast, scalable, and user-friendly tool for maximum likelihood phylogenetic inference," *bioRxiv*,.

47. Kumar S, Stecher G, Tamura K (2016), "MEGA7: Molecular evolutionary

genetics analysis version 7.0 for bigger datasets," *Molecular biology and evolution*, Vol. 33(7), pp.1870–1874.

48. Lanave C, Preparata G, Saccone C, Serio G (1984), "A new method for calculating evolutionary substitution rates," *Journal of molecular evolution*, Vol. 20(1), pp.86–93.

49. Le SV, von Haeseler A (2004), "IQPNNI: Moving fast through tree space and stopping in time," *Molecular Biology and Evolution*, Vol. 21(8), pp.1565–1571.

50. Lemey P, Salemi M, Vandamme A-M (Editors) (2009), *The phylogenetic handbook: A practical approach to phylogenetic analysis and hypothesis testing*, 2nd ed., Cambridge University Press.

51. Lemoine F, Domelevo Entfellner J-B, Wilkinson E, Correia D, Dávila Felipe M, De Oliveira T, et al. (2018), "Renewing Felsenstein's phylogenetic bootstrap in the era of big data," *Nature*, Vol. 556(7702), pp.452–456.

52. Lewis PO, Holder MT, Holsinger KE (2005), "Polytomies and Bayesian phylogenetic inference," *Systematic Biology*, Vol. 54(2), pp.241–253.

53. van der Linde K, Houle D, Spicer GS, Steppan SJ (2010), "A supermatrix-based molecular phylogeny of the family Drosophilidae," *Genetics Research*, Vol. 92(1), pp.25–38.

54. Lodish H, Berk A, Kaiser CA, Krieger M, Bretscher A, Ploegh H, et al. (2013), *Molecular cell biology*, 7th ed., New York: W.H. Freeman and Co.

55. Minh BQ, Hahn M, Lanfear R (2018), "New methods to calculate concordance factors for phylogenomic datasets," *bioRxiv*.

56. Minh BQ, Nguyen MAT, von Haeseler A (2013), "Ultrafast approximation for phylogenetic bootstrap," *Molecular Biology and Evolution*, Vol. 30(5), pp.1188–1195.

57. Minh BQ, Vinh LS, von Haeseler A, Schmidt HA (2005), “pIQPNNI: parallel reconstruction of large maximum likelihood phylogenies,” *Bioinformatics*, Vol. 21(19), pp.3794–3796.

58. Nesse RM, Williams GC (2012), *Why we get sick: The new science of Darwinian medicine*, 1st ed., Vintage.

59. Nguyen L-T, Schmidt HA, von Haeseler A, Minh BQ (2015), “IQ-TREE: a fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies,” *Molecular Biology and Evolution*, Vol. 32(1), pp.268–274.

60. Nguyen MAT, Klaere S, von Haeseler A (2011), “MISFITS: Evaluating the goodness of fit between a phylogenetic model and an alignment,” *Molecular Biology and Evolution*, Vol. 28(1), pp.143–152.

61. Nixon KC (1999), “The parsimony ratchet, a new method for rapid parsimony analysis,” *Cladistics*, Vol. 15(4), pp.407–414.

62. Nyakatura K, Bininda-Emonds ORP (2012), “Updating the evolutionary history of Carnivora (Mammalia): a new species-level supertree complete with divergence time estimates,” *BMC Biology*, Vol. 10(1), p.12.

63. Olsen GJ, Matsuda H, Hagstrom R, Overbeek R (1994 Feb), “fastDNAmL: a tool for construction of phylogenetic trees of DNA sequences using maximum likelihood,” *Computer applications in the biosciences : CABIOS*, Vol. 10(1), pp.41–48.

64. Ou C-Y, Ciesielski CA, Myers G, Bandea CI, Luo C-C, Korber BTM, et al. (1992), “Molecular epidemiology of HIV transmission in a dental practice,” *Science*, Vol. 256(5060), pp.1165–1171.

65. Pattengale ND, Alipour M, Bininda-Emonds ORP, Moret BME, Stamatakis A (2010 Mar), “How many bootstrap replicates are necessary?,” *Journal of computational biology : a journal of computational molecular cell biology*,

Vol. 17(3), pp.337–354.

66. Pohl A, Cosenza B, Mesa MA, Chi CC, Juurlink B (2016), “An evaluation of current SIMD programming models for C++,” In: Proceedings of the 3rd Workshop on Programming Models for SIMD/Vector Processing, New York, NY, USA: ACM, pp.3:1--3:8.

67. Posada D, Crandall KA (1998), “MODELTEST: testing the model of DNA substitution,” *Bioinformatics (Oxford, England)*, Vol. 14(9), pp.817–818.

68. Pyron RA, Burbrink FT, Colli GR, de Oca ANM, Vitt LJ, Kuczynski CA, et al. (2011), “The phylogeny of advanced snakes (Colubroidea), with discovery of a new subfamily and comparison of support methods for likelihood trees,” *Molecular Phylogenetics and Evolution*, Vol. 58(2), pp.329–342.

69. Rambaut A, Grassly NC (1997), “Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees,” *Bioinformatics*, Vol. 13(3), pp.235–238.

70. Ronquist F, Teslenko M, van der Mark P, Ayres DL, Darling A, Höhna S, et al. (2012), “MrBayes 3.2: Efficient Bayesian phylogenetic inference and model choice across a large model space,” *Systematic Biology*, Vol. 61(3), pp.539–542.

71. Saitou N, Nei M (1987), “The neighbor-joining method: a new method for reconstructing phylogenetic trees,” *Molecular Biology and Evolution*, Vol. 4(4), pp.406–425.

72. Salichos L, Rokas A (2013 May 16), “Inferring ancient divergences requires genes with strong phylogenetic signals,” *Nature*, Vol. 497(7449), pp.327–331.

73. Sanderson MJ, Donoghue MJ, Piel WH, Eriksson T (1994), “TreeBASE: a prototype database of phylogenetic analyses and an interactive tool for browsing the phylogeny of life,” *American Journal of Botany*, Vol. 81(6),

p.183.

74. Sankoff D (1975), “Minimal mutation trees of sequences,” *SIAM Journal on Applied Mathematics*, Vol. 28(1), pp.35–42.

75. Schmidt HA, Strimmer K, Vingron M, von Haeseler A (2002), “TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing,” *Bioinformatics*, Vol. 18(3), pp.502–504.

76. Simmons MP, Norton AP (2014), “Divergent maximum-likelihood-branch-support values for polytomies,” *Molecular Phylogenetics and Evolution*, Vol. 73, pp.87–96.

77. Springer MS, Meredith RW, Gatesy J, Emerling CA, Park J, Rabosky DL, et al. (2012 Nov 16), “Macroevolutionary dynamics and historical biogeography of primate diversification inferred from a species supermatrix,” Stanyon R, editor. *PLoS ONE*, Vol. 7(11), p.e49521.

78. Stamatakis A, Ludwig T, Meier H (2005), “RAxML-III: a fast program for maximum likelihood-based inference of large phylogenetic trees,” *Bioinformatics*, Vol. 21(4), pp.456–463.

79. Stamatakis A (2006), “RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models,” *Bioinformatics*, Vol. 22(21), pp.2688–2690.

80. Stamatakis A, Alachiotis N (2010), “Time and memory efficient likelihood-based tree searches on phylogenomic alignments with missing data,” *Bioinformatics*, Vol. 26(12), pp.i132-139.

81. Stamatakis A, Hoover P, Rougemont J, Renner S (2008), “A rapid bootstrap algorithm for the RAxML web servers,” *Systematic Biology*, Vol. 57(5), pp.758–771.

82. Strimmer K, Von Haeseler A (1996), “Quartet puzzling: a quartet maximum-

likelihood method for reconstructing tree topologies," *Molecular Biology and Evolution*, Vol. 13(7), pp.964–969.

83. Suzuki Y, Glazko G V, Nei M (2002), "Overcredibility of molecular phylogenies obtained by Bayesian phylogenetics," *Proceedings of the National Academy of Sciences*, Vol. 99(25), pp.16138–16143.

84. Swofford DL (2002), *PAUP*. Phylogenetic analysis using parsimony (*and other methods). Version 4*, Sunderland, Massachusetts: Sinauer Associates.

85. Tavaré S (1986), "Some probabilistic and statistical problems in the analysis of DNA sequences," In: American Mathematical Society: Lectures on Mathematics in the Life Sciences, Amer Mathematical Society, pp.57–86.

86. Telford MJ, Budd GE, Philippe HH (2015 Oct 17), "Phylogenomic Insights into Animal Evolution," *Current biology : CB*, Vol. 25(19), pp.R876-87.

87. Waddell PJ, Penny D, Moore T (1997), "Hadamard conjugations and modeling sequence evolution with unequal rates across sites," *Molecular Phylogenetics and Evolution*, Vol. 8(1), pp.33–50.

88. Wagner WH (1961), "Problems in the classification of ferns," *Recent advances in botany*, Vol. 1, pp.841–844.

89. Wang H, Wu P, Tanase IG, Serrano MJ, Moreira JE (2014), "Simple, portable and fast SIMD intrinsic programming: generic SIMD library," In: Proceedings of the 2014 Workshop on Programming Models for SIMD/Vector Processing, New York, NY, USA: ACM, pp.9–16.

90. Warnow T (2017), *Computational phylogenetics: An introduction to designing methods for phylogeny estimation*, Cambridge University Press.

91. Weiss G, von Haeseler A (2003), "Testing substitution models within a phylogenetic tree," *Molecular Biology and Evolution*, Vol. 20(4), pp.572–578.

92. Wheeler WC (1993), "Letter to the editor: the triangle inequality and character

analysis," *Molecular Biology and Evolution*, Vol. 10(3), pp.707–712.

93. Whelan S, de Bakker PIW, Quevillon E, Rodriguez N, Goldman N (2006), "PANDIT: an evolution-centric database of protein and associated nucleotide domains with inferred trees," *Nucleic Acids Research*, Vol. 34(suppl_1), pp.D327–D331.

94. Whelan S, Money D (2010), "The prevalence of multifurcations in tree-space and their implications for tree-search," *Molecular Biology and Evolution*, Vol. 27(12), pp.2674–2677.

95. Wilkinson M (1996), "Majority-rule reduced consensus trees and their use in bootstrapping," *Molecular Biology and Evolution*, Vol. 13(3), pp.437–444.

96. Yang Z (1993), "Maximum-likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites," *Molecular Biology and Evolution*, Vol. 10(6), pp.1396–1401.

97. Yang Z (1994 Jul), "Estimating the pattern of nucleotide substitution," *Journal of molecular evolution*, Vol. 39(1), pp.105–111.

98. Yang Z (1994 Sep), "Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: approximate methods," *Journal of molecular evolution*, Vol. 39(3), pp.306–314.

99. Yang Z, Rannala B (1997), "Bayesian phylogenetic inference using DNA sequences: a Markov Chain Monte Carlo Method," *Molecular Biology and Evolution*, Vol. 14(7), pp.717–724.

100. Yang Z (1996), "Among-site rate variation and its impact on phylogenetic analyses," *Trends in Ecology & Evolution*, Vol. 11(9), pp.367–372.

101. Yang Z (2006), *Computational molecular evolution*, Oxford: Oxford University Press.

102. Yang Z, Nielsen R, Goldman N, Pedersen A-MK (2000), "Codon-substitution

models for heterogeneous selection pressure at amino acid sites," *Genetics*, Vol. 155(1), pp.431–449.

103. Yang Z, Rannala B (2012), "Molecular phylogenetics: principles and practice," *Nature Reviews Genetics*, Vol. 13(5), pp.303–314.

104. Zhou X, Shen X-X, Hittinger CT, Rokas A (2017), "Evaluating fast maximum likelihood-based phylogenetic programs using empirical phylogenomic data sets," *Molecular Biology and Evolution*, Vol. 35(2), pp.486–503.

105. Zwickl DJ (2006), "Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion," PhD Thesis, The University of Texas at Austin.

PHỤ LỤC 1: BẢNG BỔ SUNG

Bảng P1. Các dòng lệnh dùng để chạy các thuật toán của IQ-TREE và RAxML dùng trong Chương 2 luận án.

Phương pháp	Phiên bản IQ-TREE	Dòng lệnh ví dụ
UFBoot gốc	IQ-TREE 0.9.6	iqtree -s example.phy -m GTR+G -bb 1000 -p 0.5
UFBoot2	IQ-TREE 1.6.beta5	iqtree -s example.phy -m GTR+G -bb 1000
UFBoot2+NNI	IQ-TREE 1.6.beta5	iqtree -s example.phy -m GTR+G -bb 1000 -bnni
SBS	IQ-TREE 1.6.beta5	iqtree -s example.phy -m GTR+G -b 100
RAxML search	RAxML 8.2.9	raxmlHPC-SSE3 -f d -m GTRGAMMA -p $RANDOM -s example.phy -n raxsearch.example.phy
RAxML rapid bootstrap with bootstopping	RAxML 8.2.9	raxmlHPC-SSE3 -s example.phy -m GTRGAMMA -n rbs.example.phy -x $RANDOM -N autoMRE -p $RANDOM

PHỤ LỤC 2: CÁC CÂU LỆNH TNT VÀ PAUP*

1. Script TNT để thực hiện fast-TNT với ma trận chi phí đều

Phân tích bootstrap này tạo ra 1000 bản sao bootstrap và sử dụng chiến lược tìm kiếm nhanh trên cả sắp hàng gốc và các sắp hàng bootstrap.

Trên máy Linux, để thực hiện phân tích này trên một sắp hàng DNA ví dụ example.fa có định dạng fasta, hãy lưu tập lệnh dưới đây thành tệp fastboot.run, đặt nó trong cùng thư mục với tệp thực thi tnt, sau đó chạy dòng lệnh:

./tnt fastboot example.fa dna,

Với dữ liệu protein, thay thế xâu "dna" trên dòng lệnh trên thành "prot".

Khi chương trình thực hiện xong, bạn sẽ thấy ba tệp mới.

- example.fa.fast.log chứa các thông báo chương trình trong quá trình chạy.
- example.fa.fast.best chứa cây tốt nhất xây dựng cho sắp hàng gốc.
- example.fa.fast.boottrees chứa 2 cây đồng thuận và 1000 cây bootstrap.

```
macro=;
mxram 1000;
taxname +100;
taxname=;
log %1.fast.log;
collapse 0;
report =;
watch =;
nstates %2;
nstates nogaps;
p &%1;
hold 2000;
rseed 0;

mult= rep 1 hold 1;
export - %1.fast.best;

resample boot rep 1000 freq savetrees [mult = rep 1 hold 1;];
export - %1.fast.boottrees;
```

```
log/;
proc/;
z;
```

2. Script TNT để thực hiện intensive-TNT với ma trận chi phí đều

Phân tích bootstrap này tạo ra 1000 bản sao bootstrap và sử dụng chiến lược tìm kiếm kĩ trên sắp hàng gốc, chiến lược tìm kiếm nhanh cho các sắp hàng bootstrap.

Tương tự cách sử dụng fast-TNT, lưu tập lệnh sau thành tệp intensiveboot.run, sau đó chạy dòng lệnh:

./tnt intensiveboot example.fa dna,

```
macro=;
log %1.intensive.log;
report =;
watch =;
nstates %2;
nstates nogaps;
mxram 1000;
taxname +100;
taxname=;
proc &%1; sect: slack 100;
hold 10000;
rseed 0;
collapse 0;

xmult = notarget hits 3 level 0 chklevel +1 1;
export - %1.intensive.best;

resample boot rep 1000 freq savetrees [mult=rep 1 hold 1];
export - %1.intensive.boottrees;

log/;
proc/;
z;
```

3. Các lệnh TNT làm việc với ma trận chi phí không đều

Để thực hiện phân tích bootstrap cho DNA dùng ma trận chi phí không đều với fast-TNT hoặc intensive-TNT, hãy chèn vào dòng trống đầu tiên trong tập lệnh tương ứng

cho ma trận chi phí đều (đã cho ở phần trước) các lệnh sau để định nghĩa ma trận chi phí:

```
smatrix =1 (g1ts1tv2) a/c 2 a/g 1 a/t 2 c/g 2 c/t 1 g/t 2;
smatrix +1 .;
ccode -( .;
```

Với dữ liệu protein, chúng tôi sử dụng ma trận sau đây:

```
smatrix =1 (aa_nt_changes) A/R 2 A/N 2 A/D 1 A/C 2 A/Q 2 A/E 1 A/G 1
A/H 2 A/I 2 A/L 2 A/K 2 A/M 2 A/F 2 A/P 1 A/S 1 A/T 1 A/W 2 A/Y 2 A/V
1 R/N 2 R/D 2 R/C 1 R/Q 1 R/E 2 R/G 1 R/H 1 R/I 1 R/L 1 R/K 1 R/M 1
R/F 2 R/P 1 R/S 1 R/T 1 R/W 1 R/Y 2 R/V 2 N/D 1 N/C 2 N/Q 2 N/E 2 N/G
2 N/H 1 N/I 1 N/L 2 N/K 1 N/M 2 N/F 2 N/P 2 N/S 1 N/T 1 N/W 2 N/Y 1
N/V 2 D/C 2 D/Q 2 D/E 1 D/G 1 D/H 1 D/I 2 D/L 2 D/K 2 D/M 2 D/F 2 D/P
2 D/S 2 D/T 2 D/W 2 D/Y 1 D/V 1 C/Q 2 C/E 2 C/G 1 C/H 2 C/I 2 C/L 2
C/K 2 C/M 2 C/F 1 C/P 2 C/S 1 C/T 2 C/W 1 C/Y 1 C/V 2 Q/E 1 Q/G 2 Q/H
1 Q/I 2 Q/L 1 Q/K 1 Q/M 2 Q/F 2 Q/P 1 Q/S 2 Q/T 2 Q/W 2 Q/Y 2 Q/V 2
E/G 1 E/H 2 E/I 2 E/L 2 E/K 1 E/M 2 E/F 2 E/P 2 E/S 2 E/T 2 E/W 2 E/Y
2 E/V 1 G/H 2 G/I 2 G/L 2 G/K 2 G/M 2 G/F 2 G/P 2 G/S 1 G/T 2 G/W 1
G/Y 2 G/V 1 H/I 2 H/L 1 H/K 2 H/M 2 H/F 2 H/P 1 H/S 2 H/T 2 H/W 2 H/Y
1 H/V 2 I/L 1 I/K 1 I/M 1 I/F 1 I/P 2 I/S 1 I/T 1 I/W 2 I/Y 2 I/V 1
L/K 2 L/M 1 L/F 1 L/P 1 L/S 1 L/T 2 L/W 1 L/Y 2 L/V 1 K/M 1 K/F 2 K/P
2 K/S 2 K/T 1 K/W 2 K/Y 2 K/V 2 M/F 2 M/P 2 M/S 2 M/T 1 M/W 2 M/Y 3
M/V 1 F/P 2 F/S 1 F/T 2 F/W 2 F/Y 1 F/V 1 P/S 1 P/T 1 P/W 2 P/Y 2 P/V
2 S/T 1 S/W 1 S/Y 1 S/V 2 T/W 2 T/Y 2 T/V 2 W/Y 2 W/V 2 Y/V 2 ;
smatrix +1 .;
ccode -( .;
```

4. Lệnh bootstrap trong PAUP* sử dụng chiến lược giống fast-TNT với ma trận chi phí đều

```
bootstrap nreps=1000 search=heuristic format=Phylip
replace=yes/ addseq=random nreps=1 swap=tbr multrees=no
hold=1 reconlimit=Infinity
```